

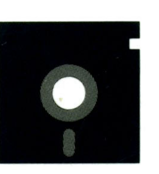
Commodore
Sachbuch

Axel Seibert

C64/C128
Spielend BASIC lernen

Eine unterhaltsame Einführung
in die Programmiersprache Basic anhand
leichtverständlicher Spielelistings

Alle im Buch vorgestellten Spiele
sind im 1541-Format auf Diskette enthalten.





Commodore Sachbuch

Axel Seibert

C64/C128 Spielend BASIC lernen

Eine unterhaltsame Einführung in die Programmiersprache Basic anhand leichtverständlicher Spielelistings

Markt&Technik Verlag AG

CIP-Titelaufnahme der Deutschen Bibliothek

Seibert, Axel:
Spielend Basic lernen / Axel Seibert. – Haar bei München: Markt-u.-Technik-Verl., 1989
ISBN 3-89090-701-6

15 14 13 12 11 10 9 8 7 6 5
92 91

ISBN 3-89090-701-6

Einbandgestaltung: Grafikdesign Heinz Rauner
Der Inhalt wurde mit dem Textverarbeitungsprogramm Word 4.0 bearbeitet
und auf der Linotronic 300 belichtet
Druck: Schoder, Gersthofen
Printed in Germany

Inhaltsverzeichnis

Einleitung

Bevor wir in diesem Buch zum eigentlichen Hauptteil, nämlich den Spielen kommen, möchte ich Ihnen einige Anmerkungen mit auf den Weg geben, die bei der Orientierung behilflich sein sollen. Zunächst möchte ich auf den Aufbau des Buches eingehen. Auf dieses einführende Kapitel folgt erst einmal ein Abschnitt, der alle Anfänger und Einsteiger mit der Materie vertraut machen soll; Sie lernen dort etwas über die Grundlagen der Basic-Programmierung auf dem C64. Sollten Sie sich angesprochen fühlen, so empfehle ich, jetzt dieses Kapitel zu überspringen, bei diesem Einsteigerteil weiterzulesen und erst danach wieder an diese Stelle zurückzukehren. Ansonsten werden die hier auftauchenden Begriffe Sie wahrscheinlich nur verwirren.

Fortgeschrittene, oder solche, die diese Grundkenntnisse sich bereits erworben haben, können nach dieser Einleitung sofort zum Spieleteil gehen; die Spiele haben keine thematische Ordnung, sie sind also nicht nach Gattungskriterien wie Action- oder Geschicklichkeitsspiel sortiert. Es handelt sich durchwegs um einfache und dennoch ansprechende Spiele; dabei wurde auf besonders problematische Bereiche wie HiRes-Grafik, Sprites und Musik verzichtet. Der interessierte Leser kann sich aber anhand der Beispiele »Snoopy« und »Lissajous« über diese Programmierung informieren.

Die einzelnen Kapitel haben einem speziellen Aufbau: Nach einer kurzen Spielbeschreibung folgt das eigentliche Spiel; daran schließt sich eine detaillierte Beschreibung der Programmstruktur sowie der neuen Befehle an.

Bei den Programmen wurde auf übersichtliche und strukturierte Programmierung geachtet. Ich möchte den prinzipiellen Aufbau an einem extremen Kurzbeispiel verdeutlichen:

Keine Panik, wenn Sie beim Betrachten dieses Beispiels nur Bahnhof verstehen. Das wird sich bald ändern. Dafür haben Sie ja dieses Buch! In den Programmen wurde Wert auf Übersichtlichkeit gelegt. Daher werden nur dort, wo es wirklich sinnvoll bzw. notwendig ist, mehrere Befehle in einer Zeile zusammengefaßt.

```
10 REM ************************
20 REM ***                  ***
30 REM *** TITEL DES SPIELES ***
40 REM ***                  ***
50 REM ************************
60 REM
70 GOSUB 1000
80 GOSUB 2000
90 GOSUB 3000
100 END
1000 REM
1010 REM *** VORBEREITUNGEN ***
1020 REM
```

```
1030 RETURN
2000 REM
2010 REM *** SPIEL ***
2020 REM
2030 RETURN
3000 REM
3010 REM *** ENDE ***
3020 REM
3030 RETURN
```

Dem Buch liegt eine Programmdiskette bei, auf der alle hier beschriebenen Spiele enthalten sind. Sie müssen die Programme also nicht abtippen. Dennoch wurden die Listings mitabgedruckt, damit alle Leser in den Genuß der Spielelistings kommen, und nicht nur die, die einen Drucker besitzen. Dennoch wurden alle Steuerzeichen in Klartext umgesetzt. (Sollten Sie nicht wissen, worum es bei Steuerzeichen geht, so möchte ich Sie bitten, im Kapitel »Der richtige Einstieg« nachzulesen.) Dadurch wird das Lesen der Listings erheblich vereinfacht. Diese Art der Darstellung hat sich außerdem in allen Zeitschriften und Büchern durchgesetzt.

Innerhalb des Buches wird eine spezielle Schreibweise der Eingaben eingehalten: Wenn Sie eine Taste drücken sollen, wird dies im Text folgendermaßen dargestellt:

[TASTE] zum Beispiel: [J], [F1] oder [5].

Längere Eingaben werden zusammenhängend groß geschrieben, zum Beispiel:

```
LOAD "SUPERHIRN",8.
```

Wenn im Text [RETURN] erscheint, dürfen Sie nicht die einzelnen Tasten [R], [E], [T], [U], [R] und [N] drücken. Denn es ist die Taste mit der Aufschrift [RETURN] gemeint.

Auch die Sondertasten werden wie herkömmliche Tasten geschrieben. Zum Beispiel stellt [CBM] die Taste links unten auf der Tastatur mit dem Commodore-Emblem (Commodore Business Machines) dar; daher der Name CBM. Bei den Cursor-Tasten werden folgende Ausdrücke verwendet:

[CRSR-UP]	Cursor nach oben
[CRSR-DOWN]	Cursor nach unten
[CRSR-RIGHT]	Cursor nach rechts
[CRSR-LEFT]	Cursor nach links

Sollen zwei Tasten gleichzeitig gedrückt werden, sind sie durch ein Plus-Zeichen (+) verbunden, zum Beispiel: [SHIFT]+[CLR/HOME] oder [CTRL]+[9]. Dagegen werden zwei aufeinanderfolgende Tastendrücke durch ein Leerzeichen getrennt:

[J] [RETURN]

Zu der beigelegten Diskette ist noch zu sagen, daß Sie sich Sicherheitskopien für Archivierungszwecke anfertigen dürfen. Da jedes einzelne Programm aber einen Copyrightschutz genießt, dürfen Sie Ihre Kopien nicht weitergeben.

Zum Schluß noch ein Tip zu den Bildschirmfarben des C64: Sollten Sie einen Fernseher an den Computer angeschlossen haben, kann es sein, daß Sie die Schrift etwas schlecht lesen können. Drücken Sie doch einmal [CTRL]+[2]. Die Schrift sollte jetzt weiß und damit wesentlich besser zu lesen sein. Haben Sie dagegen einen Grünmonitor, so kann ich Ihnen folgende Befehle empfehlen:

```
POKE 53280,0 [RETURN]
POKE 53281,0 [RETURN]
[CBM]+[5]
```

Ansonsten rate ich, unten stehendes Programm laufen zu lassen. Wenn Ihnen eine Farbkombination gefällt, drücken Sie [RUN/STOP]. Tippen Sie nach der Verwendung des Programms ein:

```
PRINT A,B
```

Diese beiden Werte sollten Sie sich merken und nach dem nächsten Einschalten Ihres C64 folgende Zeilen eingeben:

```
POKE 53280,A [RETURN]
POKE 53281,B [RETURN]
```

Bei diesen Zeilen müssen Sie für »A« und »B« die beiden Werte einsetzen, die Sie sich beim ersten Mal für »A« und »B« merken sollten. Die passende Zeichenfarbe müssen Sie sich allerdings noch mit den Tastenkombinationen von [CTRL] bzw. [CBM] und einer der Tasten von [1] bis [8] heraussuchen.

```
10 PRINT CHR$(147)
20 FOR A=0 TO 15
30 :POKE 53280,A
40 :FOR B=0 TO 15
50 : POKE 53281,B
60 : FOR C=1 TO 1500:NEXT C
70 :NEXT B
80 NEXT A
90 END
```

Und jetzt wünsche ich viel Vergnügen und viel Erfolg beim Basic-Lernen!

Der richtige Einstieg

Ich möchte Sie an dieser Stelle erst einmal herzlich willkommen heißen. Dieses Buch wurde geschrieben, um Ihnen die Programmierung Ihres C64 nahezubringen. Da Sie als völliger Neuling auf dem Gebiet der »Computerei« noch wenige Kenntnisse haben (keine Angst, das ändert sich in nächster Zeit), müssen wir zu Beginn dieses Buches erst einmal die nötigen Grundlagen schaffen. Dazu gehört auch, daß Sie über einige Grundbegriffe Bescheid wissen. Ich möchte hier aber nur auf die Fachausdrücke eingehen, die wir in diesem Buch benötigen werden. Die übrigen Begriffe und Ausdrücke werden Sie sich im Laufe Ihrer »Programmierer-Karriere« erwerben.

Ich gehe davon aus, daß Sie bereits eine voll funktionsfähige Computeranlage, bestehend aus einem C64(C128), einem Diskettenlaufwerk und einem Bildschirm, vor sich stehen haben. Ich empfehle Ihnen, Ihren Computer einzuschalten und sich mit diesem Buch vor den Computer zu setzen. So dürfte es am leichtesten sein, alle Schritte gleich nachzuvollziehen.

Auf der Gehäuseoberseite sehen Sie eine ganz einfache Schreibmaschinentastatur. Wann immer Sie Ihrem C64 etwas mitteilen wollen, geschieht dies über diese Tastatur. Sie funktioniert genauso wie die einer Schreibmaschine. Zugegeben, die Tasten sind teilweise anders beschriftet, aber im Prinzip ...

Wenn Sie wollen, können Sie jetzt einfach einmal wild darauf lostippen. Sie sehen, auf dem Bildschirm ein kleines blinkendes Quadrat, das über den Bildschirm wandert und immer Ihre gedrückten Tasten ausgibt. Dieses Quadrat nennt man »Cursor«. Mit ihm können Sie an alle Positionen auf dem Bildschirm fahren. Wie, das lernen Sie erst später.

Drücken Sie jetzt die Taste mit der Aufschrift [RETURN]. Sollten in derselben Zeile links vom Cursor noch Zeichen stehen, bevor Sie [RETURN] gedrückt haben, wird der Computer die Meldung »`?SYNTAX ERROR`« und »`READY.`« ausgeben. Lassen Sie sich durch diese erste Reaktion nicht verunsichern. Drücken Sie noch mehrere Male diese [RETURN]-Taste. Sie sehen, wie der Cursor pro Tastendruck um eine Zeile nach unten wandert. Wenn Sie oft genug diese Taste betätigen, wird bald die ursprüngliche Meldung Ihres Systems »`**** COMMODORE 64 BASIC V2 ****`« etc. vom Bildschirm verschwinden, sie wird nach oben verschoben, in Computer-Latein: gescrollt. Den Vorgang des Verschiebens nennt man auch »Scrolling«.

Auf alle Fälle befindet sich Ihr Cursor jetzt in der linken unteren Ecke des Bildschirms. Wenn Sie ihn aber wieder links oben haben wollen, drücken Sie einmal die Taste mit der Aufschrift [CLR/HOME] in der rechten oberen Ecke Ihrer Tastatur.

Doch es gibt noch mehr Tasten, die Sie von einer Schreibmaschine her nicht kennen. Da wären zum Beispiel die sogenannten »Cursor-Tasten«. Sie befinden sich rechts unten auf der Tastatur und sind mit dem Wort »CRSR« und jeweils zwei Pfeilen beschriftet. Mit diesen beiden Tasten können Sie den Cursor an jede beliebige Stelle auf dem Bildschirm fahren. Da sich Ihr Cursor im Moment in der linken oberen Ecke befindet, drücken Sie einmal die linke der beiden Cursor-Tasten. Ihr Cursor bewegt sich jetzt um eine Zeile nach unten. Möchten Sie ihn dagegen nach oben »drücken«, müssen Sie gleichzeitig die sogenannte [SHIFT]-Taste und diese linke Cursor-Taste drücken.

Erkennen Sie den Zusammenhang zwischen Tastenaufschrift und Auswirkung? Sobald Sie nur diese linke Cursor-Taste drücken, wird der Cursor nach unten bewegt (entspricht dem Pfeil nach unten). Drücken Sie dagegen zusätzlich die [SHIFT]-Taste, wird der Pfeil nach oben verwendet. Das gilt grundsätzlich für alle Tasten, die doppelt beschriftet sind (wie zum Beispiel [CLR/HOME], [INST/DEL] und [RUN/STOP]): Wenn Sie nur diese Taste drücken, wird die untere Aufschrift ausgeführt; sobald Sie aber gleichzeitig noch die [SHIFT]-Taste drücken, wird die obere Bedeutung verwendet.

Dasselbe gilt natürlich auch für die zweite Cursor-Taste. Es geht ganz analog. Drücken Sie nur die rechte Cursor-Taste, bewegt sich Ihr blinkendes Quadrat nach rechts, mit [SHIFT] dagegen nach links.

Wie Sie bereits festgestellt haben, werden in diesem Buch alle Tasten besonders dargestellt, nämlich [TASTE]. Auch für die Cursor-Tasten haben wir eine ähnliche Schreibweise vorgesehen:

[CRSR-DOWN]	Cursor nach unten
[CRSR-UP]	Cursor nach oben
[CRSR-RIGHT]	Cursor nach rechts
[CRSR-LEFT]	Cursor nach links

Im Einführungskapitel finden Sie noch einmal eine ähnliche Zusammenstellung, bei der auch die restlichen Tasten mitberücksichtigt worden sind.

Kommen wir zu den restlichen »Sondertasten«: Die [CLR/HOME]-Taste haben Sie bereits benutzt, um den Cursor in die »Home«-Position (links oben) zu bringen. Wenn Sie noch [SHIFT] drücken, kommt der Cursor nicht nur nach links oben, sondern der Bildschirm wird zusätzlich noch gelöscht. Falls Sie später einmal den Überblick auf Ihrem Bildschirm verlieren sollten, drücken Sie einfach [SHIFT]+[CLR/HOME] (steht für [SHIFT] und gleichzeitiges [CLR/HOME]).

Noch ein kleiner Tip am Rande: Als Anfänger versucht man oft, die [SHIFT]-Taste und eine andere Taste wirklich zur gleichen Zeit zu drücken, das heißt man hämmert auf die Tasten ein und bemüht sich, sie zur selben Zeit zu erwischen. Das ist allerdings ein sehr mühseliges Unterfangen, da der C64 mit etlichen tausend Abfragen pro Sekunde immer etwas schneller ist als sein Benutzer. Darum mein Rat: Halten Sie die [SHIFT]-Taste

gedrückt, und tippen Sie dann erst die entsprechende Sondertaste. Dadurch wird gewährleistet, daß der Computer Ihre Eingabe auch wirklich als gleichzeitig auffaßt.

Der besten Sekretärin unterlaufen Tippfehler – warum dann nicht uns armen Computerbesitzern? Daran dachten die Computerentwickler und bauten eine Taste ein, mit deren Hilfe man auf dem Bildschirm ganz einfach Zeichen löschen und einfügen kann: die [INST/DEL]-Taste. Dabei steht »DEL« für DELETE (zu deutsch löschen) im Gegensatz zu »INST« für INSERT (zu deutsch einfügen). Ich brauche wohl nicht mehr extra zu erwähnen, daß Sie zum Löschen diese Taste verwenden können, für das Einfügen jedoch zusätzlich die [SHIFT]-Taste brauchen. Die Tasten am Rand, die sogenannten Funktionstasten von [F1] bis [F8], haben jetzt noch keine Bedeutung. Sollten sie eine erhalten, wird das im entsprechenden Kapitel näher erläutert.

Somit dürften erst einmal alle wichtigen Neuerungen abgedeckt sein, die die Tastatur mit sich brachte. Aber da gibt es noch viele, viele mehr ... Ein Computer ist dazu da, uns Arbeit abzunehmen. Dafür muß man ihm aber erst sagen, was er tun soll. Damit der Computer auch versteht, was wir ihm sagen wollen, müssen wir mit ihm in einer »Computersprache«, auch »Programmiersprache« genannt, reden. Zu diesem Zweck gibt es verschiedene Programmiersprachen wie Basic, Pascal, Modula, C, Assembler, Fortran und viele andere mehr. Im C64 ist der sogenannte »Basic-Interpreter«, der Basic-Befehle für den Computer verständlich macht, eingebaut.

Eine solche Programmiersprache ist zwar meistens an das Englische angelehnt, aber man kann keine ganzen Sätze formulieren, sondern muß sich auf einige Befehle einschränken. Diese Befehle teilen dem Computer genau mit, was er zu tun hat. Sollte Ihr C64 etwas nicht verstehen oder einen Fehler in Ihren Befehlen entdecken, wird er sich mit einer Fehlermeldung beschweren.

Doch zurück zu den Befehlen. Es gibt zwei Arten, wie Befehle verwendet werden können: im Direkt- oder im Programmodus. Im Direktmodus geben Sie einzelne Befehle ein, die sofort nach der Eingabe ausgeführt werden. Im Programmodus dagegen wartet der Computer geduldig, bis Sie alle Befehle eingegeben haben. Anschließend müssen Sie ihm mit »RUN« den Befehl mitteilen, diese Befehlsfolge, genannt Programm, abzuarbeiten.

Das Programm hat den Vorteil, daß Sie es mehrere Male laufen lassen und auch abspeichern können, während Sie im Direktmodus alle Eingaben wiederholen müßten. Wie kann der Computer diese beiden Modi unterscheiden? Wenn Sie auf dem C64 einen Befehl als Teil eines Programms, also als Programmzeile verstanden haben wollen, müssen Sie dem Befehl eine Zeilennummer voranstellen. Dabei sind Nummern von 0 bis 63999 erlaubt. Nach diesen Nummern sortiert der Computer auch Ihr komplettes Programm, daß heißt, Sie können jedes beliebige Basic-Programm um Zeilen erweitern, sofern noch Platz vorhanden ist.

Geben Sie beispielsweise folgende Zeilen ein:

```
30 PRINT "ICH WUENSCHE IHNEN EINEN GUTEN TAG."
20 PRINT "GRUESS GOTT!"
```

An dieser Stelle möchte ich auf etwas ganz Wichtiges hinweisen: Sie müssen alle Ihre Eingaben, gleichgültig, ob es ein einzelner Befehl oder eine Programmzeile ist, immer mit RETURN abschließen. Solange Sie diese Taste nicht gedrückt haben, ignoriert der Computer Ihre Eingabe.

Wenn Sie jetzt den Befehl »LIST« eingeben (RETURN nicht vergessen!), sehen Sie Ihr erstes Programm auf dem Bildschirm; die Zeilen wurden zwischenzeitlich aber sortiert.

Wie eine solche Programmzeile auszusehen hat, bleibt natürlich auch nicht Ihrer Willkür überlassen. Die Befehle, die Sie in einer Zeile verwenden dürfen, haben genau vorgeschriebene Argumente; dabei handelt es sich um weitere Angaben, die dem Computer meistens Daten übergeben, damit dieser sie beispielsweise auf dem Bildschirm darstellen soll. Oder wenn Sie in einem Programm die Sinusfunktion benutzen, so wird in der Zeile »A=SIN (X)« das »X« als das Argument bezeichnet. Es gibt zu fast allen Befehlen Argumente, die diese Befehle ziemlich flexibel halten. Wie diese zusätzlichen Angaben jeweils auszusehen haben, werde ich an Ort und Stelle erklären.

In einer Programmzeile können Sie auch mehrere Befehle zusammenfassen. Dabei müssen die einzelnen Befehle durch Doppelpunkte voneinander getrennt sein.

Von einer Programmiertechnik, bei der möglichst viele Befehle in eine Zeile gequetscht werden, kann ich nur abraten. Nach einer Woche schon verstehen Sie Ihre eigenen Programme nicht mehr. Wenn Sie dagegen jedem Befehl seine eigene Zeile widmen, werden Programme wesentlich übersichtlicher. Außerdem können Sie leichter etwaige Verbesserungen einfügen. Ich möchte natürlich nicht verheimlichen, daß die andere Methode auch Vorteile hat, nämlich geringeren Speicherplatzaufwand und höhere Verarbeitungsgeschwindigkeit. Dennoch kann ich davon nur dringend abraten.

Da wir schon bei der Programmiertechnik sind: Sehr oft können Sie mehrere Zeilen (gedanklich) zu einem geschlossenen Programmteil zusammenfassen. Vor allem, wenn derselbe Teil öfter in einem Programm vorkommt, empfiehlt es sich, diesen Teil ein für alle Mal am Ende des Programms zu plazieren; jedesmal, wenn dieser Teil im laufenden Programm (im Hauptprogramm) benötigt wird, springt man (mit Hilfe eines bestimmten Befehls) zu diesem ausgekoppelten Teil und danach wieder zurück. Trotz der grauen Theorie hoffe ich, Ihnen verständlich gemacht zu haben, wie man allgemein Unterprogramme benutzt. In den einzelnen Spielen werden Sie die Programmiertechnik noch öfters antreffen.

Am Ende dieser Ausführungen über Programmiertechniken kommen wir zu einem sehr wichtigen Kapitel, den Variablen. Vielleicht haben Sie noch nie von Variablen gehört, aber sie zählen mit zu den wichtigsten Werkzeugen eines Programmierers. In Variablen kann der Programmierer beliebige Werte speichern. Doch sobald der Computer ausgeschaltet wird, geht der Inhalt sämtlicher Variablen verloren. Auch kann man sie nicht einfach speichern wie ein Basic-Programm, sondern muß eine Routine dafür selbst schreiben.

Was Sie in einer Variablen speichern wollen, bleibt Ihnen überlassen; nur bei bestimmten Werten müssen Sie dies dem Computer vorher mitteilen.

Geben Sie zum Beispiel ein:

```
X=10
```

Ich hoffe, Sie haben selbst an [RETURN] gedacht. In Zukunft werde ich Sie aber nicht mehr daran erinnern! Mit der oben gezeigten Zeile haben Sie der Variablen »X« den Wert 10 zugewiesen. Wenn Sie sich jetzt den Inhalt der Variablen ansehen möchten, geben Sie folgendes ein:

```
PRINT X
```

Dieser Befehl gibt Variablen und anderes auf dem Bildschirm aus; mehr darüber erfahren Sie im ersten Kapitel.

Sie können »X« aber auch andere Werte zuweisen, zum Beispiel:

```
X=56.9
X=SIN(3)
```

etc. Die Variable muß auch nicht »X« heißen. Folgende Zeilen versteht Ihr C64 ebenso:

```
Y=9
Y=111.11
Y=COS(1)
```

Den Variablen können Sie (fast) beliebige Namen geben. Die Einschränkung ist folgende: Im Variablenname darf kein Basic-Befehl enthalten sein. (In Anhang A finden Sie eine Übersicht über alle Befehle.) Daher sind solche Namen wie ELISTE (enthält LIST) und DURPRINTO (enthält PRINT) nicht zulässig. Außerdem sind Namen, die länger als zwei Zeichen sind, sinnlos, da sich der C64 nur die ersten zwei Zeichen einer Variablen merkt.

Langsam wird es interessant: Sie können nämlich zwei Variablen verbinden, zum Beispiel:

```
PRINT X+Y
```

oder Sie weisen den Wert dieser Addition einer dritten Variablen zu:

```
Z=X+Y
Z=X*Y
Z=X*SIN(Y)
```

Ich glaube, an Hand dieser wenigen Beispiele wird schon klar, wie wichtig Variablen im Alltag eines Programmierers sind.

Ich sagte vorhin, Sie könnten einer Variablen nicht jeden Wert zuweisen. Zu dieser Ausnahme kommen wir jetzt: Wollen Sie in einer Variablen Buchstaben oder sonstige Zeichen (zum Beispiel Fragezeichen) speichern, muß der C64 das zuvor erfahren. Dazu hängen Sie einfach ein Dollarzeichen ($) an den Variablennamen an, zum Beispiel

```
X$
```

Auf weitere Besonderheiten werden wir in den einzelnen Kapiteln eingehen. Allmählich dürften Sie sich das nötige Grundwissen verschafft haben. Als Abschluß dieses Kapitels möchte ich noch ein paar praktische Tips und einige Anmerkungen zu bereits Gelerntem mit auf den Weg geben:

Mittlerweile haben Sie bereits den Befehl »LIST« kennengelernt, mit dessen Hilfe Sie sich Ihr Basic-Programm anzeigen ließen. Doch auch dieser Befehl kennt einige Argumente, die Sie wissen sollten, bevor Sie sich mit den Programmen beschäftigen.

So bietet dieser Befehl die Möglichkeit, sich ein Programm nur teilweise darstellen zu lassen, da man immer nur einen Ausschnitt von 25 Zeilen sehen kann. Daher kann man sich mit:

```
LIST 10 - 20
```

die Programmzeilen zwischen den Zeilen 10 und 20 anzeigen lassen. Selbstverständlich können Sie beliebige andere Zeilennummern wählen, sofern sie im gültigen Bereich (0 – 63999) liegen.

Möchten Sie ein Programm von Anfang bis zu einer bestimmten Zeile LISTen lassen, brauchen Sie bloß den Befehl etwas abzuwandeln:

```
LIST -23594
```

Und schließlich gibt es noch die Möglichkeit, sich das Programm von einer bestimmten Zeile bis zum Ende anzeigen zu lassen, und zwar mit:

```
LIST 13968-
```

Wie Sie beim Experimentieren mit dem LIST-Befehl festgestellt haben, ist die Ausgabe auf dem Bildschirm zu schnell, um eine Zeile oder einen Programmabschnitt zu überprüfen oder gar auf einen Fehler hin zu untersuchen. Daher gibt es eine Taste, mit deren Hilfe Sie die Bildschirmausgabe verlangsamen können. Sie heißt Control-Taste und befindet sich in der zweiten Reihe am linken Rand. Im Buch wird sie als [CTRL] dargestellt. Sobald Sie ein Programm LISTen und den Computer etwas bremsen möchten, brauchen Sie nur diese Taste zu drücken.

Sollte Ihnen diese Geschwindigkeit aber auch noch zu schnell sein, bzw. haben Sie einen Fehler in einer Zeile entdeckt, drücken Sie [RUN/STOP], die Taste unterhalb von [CTRL]. Dadurch wird das LISTen unterbrochen. (Übrigens können Sie mit dieser Taste auch ein normales Basic-Programm unterbrechen.) Auf dem Bildschirm erscheint dann die Meldung »`?BREAK READY.`« und Ihr Cursor. Sie können jetzt Ihr Programm beliebig editieren. Möchten Sie das LISTen allerdings fortsetzen, müssen Sie den »LIST«-Befehl neu eingeben.

Bevor Sie aber irgendein Programm LISTen können, müssen Sie eins im Speicher haben. Dafür ist die beigelegte Diskette zuständig. Auf ihr befinden sich alle in diesem Buch beschriebenen Programme. Wenn Sie nun eines dieser Spiele von der Diskette laden wollen, so geben Sie

```
LOAD "PROGRAMMNAME",8
```

ein. Anstelle von »PROGRAMMNAME« müssen Sie natürlich den Namen des Spiels einsetzen, das Sie laden möchten. Generell ist das der in der Überschrift angegebene Name; sollte das nicht der Fall sein, wird das extra vermerkt.

Was den weiteren Umgang mit externen Geräten wie der Floppy oder dem Drucker angeht, möchte ich Sie auf Ihre Handbücher und Spezialliteratur verweisen. Es würde den Rahmen dieses Buches sprengen, wollte man darauf auch noch eingehen. Sobald Sie aber ein Programm geladen haben, können Sie es mit dem Befehl

```
RUN
```

starten. Wie Sie es wieder unterbrechen können, habe ich vorhin schon gesagt [RUN/STOP]. Nach jeder Unterbrechung können Sie das Programm mit »RUN« wieder starten. (Ein letztes Mal: An [RETURN] haben Sie selbst gedacht!?)

Bevor ich Sie in die »rauhe« Basic-Welt entlasse, möchte ich noch ein paar Tips für eigene Programme mit auf den Weg geben, die Sie in den Beispielprogrammen größtenteils verwirklicht finden werden:

Numerieren Sie Ihre Programmzeilen immer so, daß ein genügend großer Abstand dazwischen für eventuelle Verbesserungen bleibt. Wenn Sie einem zehnzeiligen Programm die Zeilennummern 1 bis 10 verpassen, werden Sie sich schwer tun, da noch irgendwelche Zeilen einzufügen, es sei denn, Sie numerieren alles neu. Nehmen Sie lieber die Nummern 10 bis 100. Somit ist gesichert, daß Sie noch neun Zeilen zwischen 10 und 20 einfügen können.

Quetschen Sie nicht so viele Befehle in eine Zeile. Ein Programm ist wesentlich übersichtlicher, wenn in einer Zeile ein, zwei, maximal aber drei Befehle stehen. Sollten die Befehle sinngemäß zusammengehören (auf Beispiele werde ich in den Programmen eingehen), können diese zusammengefaßt werden. Aber lieber schreiben Sie ein paar Zeilennummern mehr und können Ihr eigenes Programm nach einer Woche noch lesen, als daß Sie an Ihrem eigenen sogenannten »Spaghetti-Code« verzweifeln.

Eine weitere Technik, die die Lesbarkeit von Programmen enorm erhöht, ist das Einrücken. Sehen Sie sich irgendein Listing an: Sobald eine Schleife auftaucht (Erklärung folgt noch), werden die folgenden Zeilen eingerückt. Meines Erachtens sind solche Programme wesentlich besser zu lesen. (Ich stehe mit dieser Meinung bestimmt nicht alleine da!)

Ein letzter Tip für alle, die mit inversen Zeichen auf dem Bildschirm zu kämpfen haben: Der C64 hat eine Eigenart, die es so auf keinem anderen Computer gibt: Sobald man ein Anführungszeichen eingegeben hat, zeigt er alle Cursor-Tasten, [CLR/HOME], [SHIFT]+[CLR/HOME] und [SHIFT]+[INST/DEL] mit inversen Zeichen an, das heißt, daß der Hintergrund dunkel und die Schrift hell wird. Erst wenn Sie noch einmal ein Anführungszeichen – [SHIFT]+[2] – eingeben, wird dieser Modus wieder beendet.

Die einzige Sondertaste, die während dieser Betriebsart funktioniert, ist die [INST/DEL]-Taste; Sie können also immer noch löschen. Allerdings bringt es nichts, das Anführungszeichen zu löschen; dennoch druckt der Computer die Sondertasten als Zeichen aus. Eine Möglichkeit, dem C64 zu entkommen, ist das Drücken der [RETURN]-Taste. Eine zweite Möglichkeit ist, wie oben bereits angedeutet, das Anführungszeichen noch einmal einzugeben. Danach können Sie wieder ganz normal weiterschreiben und auch mit den Cursor-Tasten auf dem Bildschirm herumfahren.

Wie man sich diese Eigenart in Programmen zunutze machen kann, erfahren Sie später. An dieser Stelle möchte ich aber bereits klären, wie diese reversen Zeichen in den Listings behandelt haben. Wenn Sie sich einmal die Listings durchschauen, werden Sie kein solches Zeichen sehen. Vielleicht sind Ihnen aber sonst einige Besonderheiten aufgefallen?

Betrachten Sie doch einmal im Programm »Superhirn« die Programmzeile 2030. Dort sehen Sie in geschweiften Klammern die Zeichen »CLR,2DOWN«. Ich will Ihnen das erläutern: Hätten wir die reversen Zeichen in das Buch gedruckt, so hätten Sie wahrscheinlich nichts als schwarze Flecken gesehen, da die Farbe zusammengelaufen wäre. Selbst wenn Sie die einzelnen Zeichen hätten unterscheiden können, wäre es für Sie umständlich gewesen, den Grafikzeichen ihre richtige Bedeutung zuzuweisen. Darum wurde ein Programm geschrieben, das Basic-Programme ausdruckt und dabei diese Sonderzeichen in ein lesbares Format umwandelt. Das bedeutet für Sie in der Praxis (angenommen, Sie wollen eine solche Zeile einmal in Ihren Computer eingeben), daß Sie NICHT die geschweifte Klammer eingeben (Sie würden sich schwer tun, da dieses Zeichen so auf dem C64 nicht existiert...), sondern die Taste drücken, die anschließend gezeigt wird.

Gehen wir noch einmal auf unser Beispiel, die Programmzeile 2030, ein: Sie müßten demnach eingeben: 2030 PRINT ". Jetzt drücken Sie bitte die Taste [SHIFT]+[CLR/HOME] und dann zweimal [CRSR-DOWN]; den Rest können Sie wie gewohnt eingeben. Jetzt werden Sie fragen, warum Sie die Taste [SHIFT] drücken sollten, da im Listing nur CLR stand!? Würden im Programm dieselben Abkürzungen wie in den Büchern verwendet, würden die Programmzeilen viel zu lang. Darum wurden hier einige Abkürzungen vorgenommen. Die Bedeutung der einzelnen Ausführungen werden im folgenden tabellarisch dargestellt:

CLR	[SHIFT]+[CLR/HOME]
HOME	[CLR/HOME]
DOWN	[CRSR-DOWN]
UP	[CRSR-UP]
LEFT	[CRSR-LEFT]
RIGHT	[CRSR-RIGHT]
SPACE	[SPACE]
RVSON	[CTRL]+[9]
RVOFF	[CTRL]+[0]

Wenn Buchstaben unterstrichen sind, müssen Sie gleichzeitig dazu die [SHIFT]-Taste drücken, während ein Oberstrich bedeutet, daß Sie die [CBM]-Taste dazudrücken müssen. Zahlen vor so einem Zeichen zeigen Ihnen, wie oft Sie die entsprechende Taste drücken sollen. Innerhalb einer geschweiftenKlammer können mehrere solcher Anweisungen durch Kommas getrennt aufgeführt werden. Es gibt jetzt immer noch solche Zeichen, die in dieser Liste nicht aufgeführt wurden: Das sind die Farben. Aber ich denke, Sie schaffen es, wenn im Listing ein BLACK steht, [CTRL]+[1] zu drücken. Ein Beispiel für diese Farben sehen Sie in Zeile 1060 des Spiels »Faßfarben«."

1 Superhirn

Einführung

Dieses Spiel ist zum Einsteigen ganz gut geeignet, da es relativ kurz und gut zu verstehen ist. Sicher kennen Sie alle Superhirn. Allerdings konnte keine getreue Umsetzung des Spieles aufgenommen werden, da das Programm sonst zu lang und zu schwierig geworden wäre. Daher begnügen wir uns mit einer etwas »abgespeckten« Version.

In diesem Spiel müssen Sie eine dreistellige Zahl erraten. Zuerst sucht sich der Computer diese Zahl mit Hilfe von Zufallszahlen aus. Danach sind Sie an der Reihe. Dadurch, daß es sich nur um eine dreistellige Zahl handelt, bei der zusätzlich jede Ziffer nur einmal vorkommen darf, sollte es ein leichtes sein, die Zahl zu erraten. Doch zunächst möchte ich noch erklären, wie das Programm mitteilt, welche Ziffern richtig oder falsch sind.

Eine Ziffer, die völlig falsch ist, wird mit einem »=«-Zeichen dargestellt. Eine richtige Zahl am falschen Ort wird mit dem Buchstaben »O« gekennzeichnet, und die richtige Ziffer an der richtigen Stelle schließlich wird mit einem Sternchen »*« markiert. Dabei können Sie aber nicht davon ausgehen, daß diese Zeichen an derselben Stelle stehen wie die markierten Ziffern.

Angenommen, Sie geben die Zahl 736 ein, und 6 ist die einzige richtige Ziffer in dieser Zahl, so stellt das Programm das so dar: *==. Das Sternchen kommt also nicht an die dritte Position. Stimmten Position der Zeichen und der richtigen Ziffern auch noch überein, wäre es noch viel leichter als es so schon ist, die Zahl zu erraten.

Abschließend möchte ich Ihnen noch eine Tabelle zeigen, mit deren Hilfe Sie das Schema der Zeichen leicht verstehen sollten:

Computerzahl	Ihr Tip	Bewertung
749	354	O==
281	219	*O=
345	145	**=
178	817	OOO
231	768	===
397	937	*OO

Listing zu Superhirn

```
10 REM *****************
20 REM ***           ***
30 REM *** SUPERHIRN ***
40 REM ***           ***
50 REM *****************
60 REM
70 GOSUB 1000
80 GOSUB 2000
90 GOSUB 3000
100 END
1000 REM
1010 REM *** ZAHLEN AUSSUCHEN ***
1020 REM
1030 DEF FN R(X)=INT (RND(1)*X)
1040 Z1=FN R(10)
1050 Z2=FN R(10)
1060 IF Z2=Z1 THEN GOTO 1050
1070 Z3=FN R(10)
1080 IF Z3=Z1 OR Z3=Z2 THEN GOTO 1070
1090 RETURN
2000 REM
2010 REM *** ZAHLEN ERRATEN ***
2020 REM
2030 PRINT "{CLR,2DOWN}";TAB(15);"SUPERHIRN"
2040 PRINT "{3DOWN}ES KANN LOSGEHEN. ICH HABE BEREITS DIE"
2050 PRINT "DREISTELLIGE ZAHL AUSGELOST."
2060 PRINT "BITTE DRUECKEN SIE EINE TASTE."
2070 GET A$:IF A$="" THEN GOTO 2070
2080 PRINT "{CLR}"
2090 DG=1
2100 T=0:INPUT "{HOME,2DOWN}IHR TIP BITTE ";T
2110 IF T=0 THEN GOTO 2100
2120 PRINT "{DOWN,40SPACE,UP}";
2130 IF T<1000 THEN GOTO 2160
2140 PRINT "ES GEHT HIER NUR UM DREISTELLIGE ZAHLEN!":GOTO 2100
2160 T1=INT (T/100)
2170 T2=INT ((T-T1*100)/10)
2180 T3=T-(T1*100 + T2*10)
2190 IF T1<>T2 AND T1<>T3 AND T2<>T3 THEN GOTO 2210
2200 PRINT"BITTE JEDE ZIFFER NUR EINMAL !":GOTO 2100
2210 SR=0:SF=0:RE=0
2220 IF Z1=T1 THEN SR=SR+1
2230 IF Z2=T2 THEN SR=SR+1
2240 IF Z3=T3 THEN SR=SR+1
2250 IF Z1=T2 OR Z1=T3 THEN SF=SF+1
2260 IF Z2=T1 OR Z2=T3 THEN SF=SF+1
2270 IF Z3=T1 OR Z3=T2 THEN SF=SF+1
2280 RE=3-(SR+SF)
2290 PRINT "{3DOWN}DIE AUSWERTUNG IHRES TIPS BRACHTE FOL-"
2300 PRINT "GENDES ERGEBNIS :{DOWN}"
2310 IF SR>0 THEN FOR A=1 TO SR:PRINT "*";:NEXT A
2320 IF SF>0 THEN FOR A=1 TO SF:PRINT "O";:NEXT A
2330 IF RE>0 THEN FOR A=1 TO RE:PRINT "=";:NEXT A
2340 IF SR=3 THEN PRINT "{DOWN}":RETURN
2350 DG=DG+1
2360 PRINT "{2DOWN}JETZT KOMMT DER";DG;"{LEFT}. VERSUCH !"
2370 GOTO 2100
3000 REM
3010 REM *** ENDE ***
3020 REM
3030 PRINT "JETZT HABEN SIE DIE ZAHL ERRATEN !"
```

```
3040 PRINT "SIE BENOETIGTEN ";DG;"VERSUCHE."
3050 IF DG<6 THEN PRINT "NICHT SCHLECHT !":GOTO 3080
3060 IF DG<11 THEN PRINT "DAS NAECHSTE MAL WIRD'S BESTIMMT BESSER!":GOTO
     3080
3070 PRINT "NA JA! ICH HABE SCHON BESSERE ERGEBNISSE GESEHEN !"
3080 PRINT "{2DOWN}MOECHTEN SIE NOCH EINMAL SPIELEN (J/N)?"
3090 GET AN$:IF AN$="" THEN GOTO 3090
3100 IF AN$="J" THEN RUN
3110 IF AN$="N" THEN RETURN
3120 GOTO 3090
```

■ Programmbeschreibung für Superhirn

Wie ich bereits in der Einleitung erklärt habe, folgen alle Spiele ein und demselben Aufbau. Daher werde ich in diesem ersten Programm noch einmal darauf eingehen, in allen weiteren Spielen werde ich aber auf eine Erklärung der ersten drei oder vier Befehle verzichten.

In den ersten sechs Zeilen stehen nur Bemerkungen, auf englisch »remarks«, daher der Befehl »REM«. Wann immer Sie in ein Basic-Programm einen Kommentar einbauen wollen, schreiben Sie einfach »REM« und den Text dahinter. Dieser Befehl veranlaßt den Computer, den Rest dieser Zeile zu überlesen und direkt an den Anfang der nächsten Zeile zu springen. Das heißt also, daß der C64 auch einen eventuellen Befehl nach dem »REM« überliest. Das sollten Sie nie vergessen, damit Sie sich nicht irgendwann einmal wundern, warum denn der dumme Computer nicht diesen Befehl hinter dem kurzen Kommentar ausführt.

In den nächsten drei Zeilen steht der oben erwähnte Teil, der das gesamte Programm steuert. Sie können ein Programm einfach von oben nach unten schreiben oder einen Hauptteil entwerfen, der verschiedene andere Teile aufruft. Und das ist schon der gesamte Trick bei diesem Aufbau. Die Zeilen 70 bis 90 werden praktisch zum Hauptprogramm erhoben, das drei weitere sogenannte Unterprogramme aufruft. Dabei wird der Befehl »GOSUB« verwendet. Dieser Befehl veranlaßt den Interpreter in Ihrem C64, die aktuelle Zeile zu verlassen, und zu der hinter »GOSUB« angegebenen Zeile zu springen. Dabei merkt er sich die Nummer der aktuellen Zeile und wird später wieder hierher zurückkehren (den Befehl dazu lernen Sie erst später kennen). Nehmen wir einmal an, Sie wollten in einem Programm an drei verschiedenen Stellen jedesmal denselben Text ausgeben. Dann wäre es viel zu umständlich, an allen drei Stellen, genau dieselben Befehle einzusetzen. Viel bequemer ist es, diese paar Befehle in ein Unterprogramm zu packen, dieses an das Ende Ihres Programms zu setzen, und es dann mit Hilfe des GOSUB-Befehls dreimal aufzurufen. Dabei sparen Sie sich nicht nur Tipparbeit, sondern auch einigen Speicherplatz.

So etwas nennt man strukturiertes Programmieren, denn dadurch erhält Ihr Programm eine innere Struktur, die es beispielsweise erlaubt, eine mögliche Fehlerquelle einzugrenzen. Stellen Sie sich vor, Sie hätten einen Fehler im Eingabeteil dieses Programms gefunden. In unserem Fall können Sie (später) sofort feststellen, daß der Fehler zwischen den Zeilen 1000 und 2000 liegen muß, da ab Zeile 2000 der nächste Programmteil beginnt. Auf einen

zweiten Blick können Sie sogar noch weiter gehen und sagen, daß der Fehler zwischen den Zeilen 1030 und 1090 liegen muß, weil nur in diesen Zeilen wirkliche Befehle im ersten Unterprogramm stehen.

Hätten Sie dagegen das Programm einfach von oben nach unten geschrieben, vielleicht noch mit völlig unregelmäßigen Zeilennummern, müßten Sie jetzt jede einzelne Zeile durchgehen und entweder sofort nach dem Fehler suchen oder diese Zeile erst einmal einer Einheit (falls vorhanden) zuordnen. In diesem Beispiel wirkt dieses Verfahren eher lächerlich; aber sobald Sie einmal längere Programme – oder die fremder Autoren – vor sich haben, werden Sie um eine solche Struktur froh sein.

Nach den drei GOSUBs kommt noch ein weiterer Befehl: »END«. Wie unschwer zu vermuten ist, beENDet dieser Befehl ein Programm. Dabei gibt der Computer keinerlei Meldung aus. Möchten Sie also noch einen abschließenden Kommentar ausgeben, so müssen Sie das selbst tun. Zunächst möchte ich Ihnen aber erklären, wofür dieser Befehl gut ist. Wenn man ein Programm einfach von »oben nach unten« schreibt, ist das Programmende meistens gleich mit dem Programmtextende, das heißt, die letzte Programmzeile ist meistens der Programmabschluß. In einem solchen Fall kann man sich diesen END-Befehl sparen, da der Computer von selbst aufhört, findet er keine weiteren Zeilen mehr. In unserem Fall soll das Programm aber in Zeile 100 aufhören, und es kommen noch einige Programmzeilen hinterher. Sollten Sie diesen END-Befehl vergessen, wird das Programm nach dem letzten Unterprogramm einfach weiterlaufen, das heißt, in Zeile 1000 weiterarbeiten. Um diesen Fehllauf (und unter Umständen eine Fehlermeldung) zu verhindern, wird dieser END-Befehl eingebaut. Man könnte natürlich auch nach dem letzten Unterprogramm darauf verzichten, zum Hauptprogramm zurückzukehren, das hieße wieder, daß letzte Programmzeile und Programmende zusammenfielen; das ist aber schlechter Programmierstil, und den wollen wir uns doch gar nicht erst angewöhnen!

Der nächste Befehl in Zeile 1030 ist schon ein schwerer Happen, den Sie verdauen müssen (die Zeilen 1000 bis 1020 können wir doch überspringen, oder etwa nicht?): Hier wird eine Funktion definiert – daher der Befehl »DEF FN« (define function auf gut Neuhochdeutsch). Nach diesem Befehl kommt der Name der Funktion, in diesem Fall »R«. Das »X« in den Klammern dahinter sagt dem Computer, daß in dieser Funktion eine Variable »X« verwendet werden soll. Genausogut könnten Sie ein »A«, »B« oder »XY« benutzen. Bei jedem Aufruf müssen Sie dieser Funktion einen Wert mitgeben, der dann dieser Variablen (in dem Fall »X«) zugeordnet wird. (Näheres über den Aufruf einer Funktion siehe weiter unten.) Hinter dem »=«-Zeichen steht dann die eigentliche Funktion, die das Ergebnis liefert. Hier könnte beispielsweise »X*X« oder »SIN (X)« oder ähnliches stehen. Sie müssen nur beachten, daß Sie denselben Variablennamen wie vor dem »=«-Zeichen verwenden.

In diesem Programm steht gleich eine relativ komplizierte Funktion: »INT (RND(1)*X)«. Arbeiten wir das Monstrum von links nach rechts durch: INT steht für »INTEGER«, das ist ein Variablentyp auf dem Computer. Für den Anfang reicht es, wenn Sie sich merken, daß

der C64 bei dem Befehl »INT« den Nachkommateil einer Zahl abschneidet. Beispiel: »INT (0.74356)« ergibt 0; »INT (1233.9999999)« ergibt 1233. Sie sehen also, daß dieser Befehl eine Zahl nicht rundet, sondern wirklich die Ziffern nach dem Komma (auf dem Computer ist das nun einmal der Dezimalpunkt) einfach abschneidet.

»RND« steht für random (englisch, Zufall). Diese Funktion liefert eine Zufallszahl zwischen 0 und 1. Wenn Sie eine Zufallszahl zwischen 0 und einer anderen Zahl (nennen wir sie »X«) erhalten wollen, müssen Sie »RND (1)« mit »X« multiplizieren. Die 1 in Klammern hinter dem »RND« ist nämlich nur ein Wert, mit dem der Zufallsgenerator initialisiert wird. Es würde hier zu weit führen, diese Funktion genauestens zu erklären. Wir müßten uns in die Tiefen des Betriebssystems stürzen, um den Zufallsgenerator und seine Zahlen verständlich und umfassend erklären zu können. Ich kann nur empfehlen, den Zufallsgenerator mit 1 oder einer anderen positiven Zahl zu initialisieren.

Wenn Sie die Zufallszahl mit »X« multipliziert haben, bekommen Sie nur Werte, die kleiner als »X« sind, da die 1 selbst als Zufallszahl nicht vorkommt. Möchten Sie diese Zahl aber auch noch erhalten, müssen Sie an die Formel »INT (RND(1)*X)« noch »+1« anhängen. Somit erhalten Sie alle Zufallszahlen zwischen 1 und X (inklusive). Wollen Sie dagegen die Zufallszahlen zwischen 0 und X, müssen Sie in die Formel anstelle von »*X« ein »*(X+1)« einsetzen. (Diesen Ausdruck können Sie natürlich auch bereits ausgerechnet einsetzen; Beispiel: Sie brauchen ganze Zahlen zwischen 0 und 10 [jeweils inklusive]; dann schreiben Sie entweder »... RND(1) * (10+1)«. Oder aber Sie rechnen es bereits aus: »... RND (1) * 11«. Das ist alles.)

Ein letztes Beispiel soll noch einmal verdeutlichen, wie die Zufallszahlen eingesetzt werden können. Angenommen, Sie schreiben ein Programm, das einen Würfel nachahmt, müßte Ihre Funktion wie folgt aussehen:

```
INT (RND (1)*6)+1
```

In den Zeilen 1040, 1050 und 1070 sehen Sie gleich, wie eine mit »DEF FN« definierte Funktion benutzt wird. Die Funktion selbst wird mit dem Befehl »FN« und dem Namen der Funktion (in diesem Fall »R«) aufgerufen. Einer Variablen wird der Wert der Funktion zugewiesen, wobei beim Aufruf der Funktion ein Wert (in diesem Fall 10) mitgegeben wird. Für unser Spiel brauchen wir eine dreistellige Zahl. Dafür wird die Zufallsfunktion dreimal aufgerufen. Vielleicht wundern Sie sich über den Wert 10, obwohl nur die Ziffern 0 bis 9 verwendet werden können? In der Formel für die Zufallszahlen wurde auf das »+1« verzichtet. Somit erhalten wir mit unserer Formel wirklich Zahlen zwischen 0 und 9, und nicht zwischen 1 und 10.

In den Zeilen 1060 und 1080 sehen Sie zum ersten Mal eine sogenannte Abfrage: Der »IF«-Befehl leitet sie ein. Die Syntax des kompletten IF-Befehls lautet: »IF ... THEN ...«. Hinter dem IF folgt eine Bedingung. Wenn diese Bedingung erfüllt ist, wird der Teil nach THEN abgearbeitet. Beispiel: IF 13=14 THEN PRINT "GLEICH!"

In diesem Fall ist die Bedingung (13=14) offensichtlich nicht erfüllt, also arbeitet der Computer den Teil nach THEN nicht mehr ab, sondern springt gleich an den Anfang der

nächsten Zeile. »IF 15=3*5 THEN PRINT "STIMMT!"«. (PRINT druckt Meldungen auf dem Bildschirm aus; siehe weiter unten.) In diesem Fall wird der Computer die Meldung ausgeben, da die Bedingung erfüllt ist.

Sie müssen beachten, daß der Teil hinter THEN nur noch die restliche Zeile beanspruchen darf, da eine neue Zeile für den Computer nichts mehr mit einer eventuellen vorhergehenden Abfrage zu tun hat.

In unserem Programm wird geprüft, ob die Zufallszahlen Z1, Z2 und Z3 gleich sind. Somit wird sichergestellt, daß die drei Ziffern unterschiedlich sind. Sollten sie gleich sein, so wird der Computer in die Zeilen 1050 bzw. 1070 geschickt, und zwar mit dem Befehl »GOTO«.

Es dürfte wohl einsichtig sein, daß dieser Befehl den Computer veranlaßt, ab der angegebenen Zeile weiterzuarbeiten. Das heißt, wann immer der Computer auf ein »GOTO« trifft, wird er nicht die Befehle in der folgenden Zeile oder gar noch in derselben Zeile hinter dem GOTO bearbeiten, sondern sofort in die betreffende Zeile springen. Jetzt wissen Sie auch, wofür die Zeilennummern (unter anderem) gut sind.

In Zeile 1090 sehen Sie den Befehl, der ein Unterprogramm abschließt: [RETURN]. Wenn Ihr C64 auf dieses Wort stößt, springt er dahin zurück, von wo aus das Unterprogramm aufgerufen worden ist, in unserem Fall zurück zu Zeile 70; da dort hinter dem »GOSUB« aber kein Befehl mehr steht, springt der Computer gleich eins weiter in Zeile 80 und von dort aus in das Unterprogramm ab Zeile 2000.

In diesem Unterprogramm treffen Sie auf einen Befehl, der wahrscheinlich mit am häufigsten in Basic verwendet wird, nämlich auf den »PRINT«-Befehl. Er gibt einen Text, Zahlen oder Variablen auf dem Bildschirm aus. Texte müssen in Anführungszeichen stehen (zum Beispiel PRINT "DAS IST EIN BEISPIEL"), Zahlen können so angegeben werden (zum Beispiel PRINT 12.5), und Variablen werden auch einfach hinter dem Befehl aufgeführt (zum Beispiel PRINT A,B).

Verschiedene Zeichen beeinflussen die Ausgabe selbst. Wenn Sie einfach mehrere PRINT-Befehle hintereinander ausführen lassen, nimmt jeder für sich eine neue Zeile in Anspruch. Setzen Sie dagegen einen Strichpunkt hinter ein »PRINT«, wird die nächste Ausgabe direkt hinter den letzten ausgegebenen Zeichen beginnen (zum Beispiel bewirkt »PRINT "ABC";:PRINT "DEF"« dasselbe wie »PRINT "ABCDEF"«). Ein Komma dagegen läßt die nächste Ausgabe an der zehnten, zwanzigsten oder dreißigsten Stelle in dieser Zeile beginnen. Mit diesen beiden Zeichen können Sie also einfach die Bildschirmausgabe etwas steuern.

Ein weiterer hilfreicher Befehl für die Ausgabe ist der »TAB«-Befehl. Sie können jede Zeile auf dem Bildschirm in 40 Spalten unterteilen. Mit »TAB« sprechen Sie gezielt eine dieser Spalten an (zum Beispiel PRINT TAB(15)).

Eine weitere Möglichkeit, die hier noch erwähnt werden soll, ist die Benutzung von sogenannten Steuerzeichen. Einige Tasten auf Ihrem Computer haben eine besondere Funktion

(zum Beispiel Cursor-Tasten, [CLR/HOME], [INST/DEL]). Innerhalb eines Programmes können Sie diese besonderen Funktionen auch nutzen. Wenn Sie beispielsweise den Bildschirm löschen möchten, geben Sie im Direkt-Modus einfach [SHIFT]+[CLR/HOME] ein. Wenn Sie »PRINT "« eingeben und jetzt die beiden Tasten drücken, wird der Bildschirm nicht gelöscht, sondern ein inverses Herz erscheint auf dem Bildschirm. Drücken Sie noch [SHIFT]+[2] für das zweite Anführungszeichen und dann [RETURN]. Erst jetzt wird der Bildschirm gelöscht. Und wenn Sie den Cursor innerhalb des Programms nach oben, unten, rechts oder links bewegen wollen, müssen Sie nur den PRINT-Befehl benutzen und dementsprechend oft die Cursor-Tasten drücken (siehe auch Zeilen 2030, 2040, 2080, 2100, 2120, 2290, 2300, 2340, 2360, 3080). Damit dürfte die Funktion der Zeilen 2030-2060 genügend behandelt worden sein.

Erst in Zeile 2070 kommt wieder etwas Neues: Hin und wieder muß ein Programm auf eine Eingabe des Benutzers warten. Dazu gibt es zwei Befehle, den »GET«- und den »INPUT«-Befehl. Der prinzipielle Unterschied ist, daß »GET« auf das Drücken einer Taste wartet, während »INPUT« eine beliebig lange Eingabe entgegennimmt, die mit [RETURN] abgeschlossen werden muß. (Beliebig lang ist übertrieben: Die Eingabe muß kürzer als 256 Zeichen sein.)

Bei »GET« muß aber dazugesagt werden, daß der Befehl nicht richtig wartet. Man muß »Leereingaben« selbständig abfangen, wie Sie das in Zeile 2070 erkennen können. Ansonsten rast das Programm weiter, ohne daß der Benutzer gemerkt hätte, daß da ein »GET«-Befehl vorhanden ist. Ihr C64 rennt also ständig in der Zeile 2070 umher, bis Sie endlich eine Taste drücken; denn davor ist die Bedingung »A$=""« noch erfüllt (sonst hätte »A$« den Wert der Taste, die Sie gedrückt haben), und somit wird ständig der »GOTO«-Befehl abgearbeitet, der den Computer wieder zur Tastaturabfrage mittels »GET« führt.

In Zukunft werde ich Zeilen mit bereits behandelten Befehlen großzügig übergehen, es sei denn, daß etwas Neues auftaucht, oder aber die Zeilen für das Verständnis des Programms unerläßlich sind. Hier also zum letzten Mal: Mit einem inversen Herz nach PRINT wird der Bildschirm gelöscht.

In Zeile 2090 wird eine neue Variable namens »DG« (wie DurchGang) angelegt. Sie soll jede Runde mitzählen, so daß am Spielende »DG« die Zahl aller Spielrunden enthält. Grundsätzlich ist zu sagen, daß Variablen im C64 nicht besonders aufgeführt werden müssen, bevor sie das erste Mal verwendet werden. Es gibt Computersprachen (wie C, Modula), die das extra verlangen. Auch müssen die Variablen nicht alle mit Null vorbelegt werden; diese Aufgaben erledigt für uns der C64. Sie können also jederzeit neue Variablen einführen und davon ausgehen, daß sie nach dem Programmstart den Wert Null enthalten. Nur wenn Sie der Variablen einen anderen Startwert (zum Beispiel DG=1) zuweisen wollen, müssen Sie das explizit programmieren.

Dasselbe gilt für Strings (das sind Variablen, die Zeichenketten aufnehmen; um sie von »normalen« Variablen zu unterscheiden, sind sie mit einem »$«-Zeichen versehen, zum

Beispiel T$="GRUESS GOTT"). Nach dem Programmstart haben sie alle die Länge Null (entspricht der Zuweisung T$="").

In unserem Programm wird in Zeile 2100 noch die Variable »T« initialisiert (mit einem Wert belegt), weil das Programm öfters an diese Stelle springt, und »T« dann einen anderen unerwünschten Wert haben könnte.

In derselben Zeile sehen Sie auch noch das Gegenstück zu »GET«, nämlich »INPUT«. Dieser Befehl erlaubt auch die Ausgabe eines Textes vor der eigentlichen Eingabe. Dabei wird nach dem Text noch ein Fragezeichen ausgegeben. Auch wenn Sie den Befehl ohne Text anwenden, wird ein Fragezeichen auf dem Bildschirm erscheinen.

Die Syntax für diesen Befehl sieht so aus: INPUT ["BELIEBIGER TEXT";] VARIABLE. Sie sehen also, daß Sie noch einen Strichpunkt zwischen den Text und die Variable setzen müssen, der die Eingabe zugeordnet werden soll. Ansonsten kann der Variablenname sofort auf das »INPUT« folgen (Beispiel: INPUT »WIE GEFAELLT IHNEN DIESES SPIEL";A$: INPUT B$).

Ich rate Ihnen, erst ein bißchen mit den beiden Befehlen herumzuexperimentieren, bevor Sie in eigenen Programmen diese Befehle verwenden. Nur so können Sie abschätzen, welcher Befehl in einer Situation der bessere ist. In Zeile 2110 wird eine eventuelle Leereingabe abgefangen; wenig entscheidungsfreudige Menschen können sich hier also nicht vor einer Eingabe drücken!

Sollten Sie bei einer INPUT-Frage nur die [RETURN]-Taste drücken, ohne vorher eine »Antwort« eingegeben zu haben, so behält die Variable ihren ursprünglichen Wert bei. In unserem Fall bleibt T=0 erhalten, da wir vorher der Variablen (Zeile 2100) diesen Wert zugewiesen haben; daher können wir also sicher sein, daß sie bei einer eventuellen »Leereingabe« weiterhin 0 bleibt. Auf dieser Voraussetzung baut die Abfrage in Zeile 2110 auf.

In Zeile 2120 wird eine Zeile auf dem Bildschirm übersprungen und die zweite Zeile unter der Eingabezeile gelöscht. Wie Sie weiter unten noch sehen werden, benutzt das Programm immer wieder denselben Bildschirmaufbau, das heißt, der Bildschirm wird nicht gelöscht. Und da es mehrere Meldungen des Programms gibt, die auch noch unterschiedlich lang sind, könnte es nach einiger Spielzeit zu einem etwas chaotischen Bildschirmaufbau kommen. Um dies zu vermeiden, wird hier eine Zeile auf dem Bildschirm mit Leerstellen überschrieben, sprich gelöscht. Am Ende sehen Sie noch den Ausdruck [CRSR-UP], der für die Tastenkombination [SHIFT] + [LINKE CURSORTASTE] steht; dadurch werden eventuell folgende Meldungen auch wirklich in die eben gelöschte Zeile und nicht darunter gedruckt werden.

In den folgenden beiden Zeilen geht es erst einmal um die Überprüfung Ihrer Eingabe. Ihre Zahl wurde vom »INPUT«-Befehl an die Variable »T« übergeben. In Zeile 2130 wird überprüft, ob Ihre Zahl im erlaubten Bereich ist; dabei wird allerdings nur darauf geachtet, daß keine zu großen Zahlen verwendet werden. Das heißt, rein theoretisch könnten Sie »unbemerkt« negative Zahlen eingeben, aber na ja ...

Sollte Ihre Eingabe zu groß sein (das heißt T<1000 nicht erfüllt) wird das GOTO hinter der Abfrage nicht, dafür aber die Befehle in Zeile 2140 ausgeführt. Das führt zu einer entsprechenden Meldung und zu einem Sprung nach Zeile 2100, sprich zur Eingabe.

Die Zeilen 2160 bis 2200 übernehmen die Auswertung und noch einmal eine Überprüfung der eingegebenen Zahl: Die drei folgenden Zeilen ordnen den Variablen »T1«, »T2« und »T3« jeweils die Hunderter-, die Zehner- und die Einerstelle Ihrer Zahl zu. Diese Berechnungen sind allgemein gültig, also nicht computerspezifisch, und die INT-Funktion haben wir auch schon besprochen. Lediglich der Schrägstrich ist vielleicht für manche unter Ihnen noch neu; er bedeutet aber nichts anderes als Division.

Diese theoretischen Ausführungen sollen noch an einem Beispiel verdeutlicht werden: Angenommen, Sie haben 538 eingegeben; dann enthält also die Variable »T« (von Zeile 2100) diesen Wert 538. »T1« wird der ganzzahlige Wert von T / 100 zugewiesen, also:

```
T1=INT (T/100)=INT (538/100)=INT (5.38)=5
```

Mit T2 wird ähnlich verfahren:

```
T2=INT ((T-T1*100)/10)=INT ((538-5*100)/10)=INT (38/10)=3
```

Und T3 ist dann ganz leicht auszurechnen:

```
T3=T-(T1*100 + T2 * 10)=T-(500 + 30)=8
```

Nachdem Ihre Eingabe in die einzelnen Ziffern aufgeteilt worden ist, werden sie wieder überprüft. Diesmal wird getestet, ob die Ziffern alle unterschiedlich sind. Dafür wurde die Abfrage allerdings etwas anders programmiert als in den Zeilen 1060 und 1080. In 1080 wurde geprüft, ob »Z3« und »Z1« *oder* »Z3« und »Z2« identisch sind. Das heißt, daß die Bedingung in Zeile 1080 als erfüllt gilt, wenn Z3=Z1, Z3=Z2 oder gar Z3=Z2=Z1 sind.

In Zeile 2190 aber ist die Bedingung nur dann erfüllt, wenn wirklich alle Zahlen unterschiedlich sind. Diese Bedingung kann man mit Hilfe von *und* programmieren. Es müssen »T1« *und* »T2« und »T1« *und* »T3« und »T2« *und* »T3« unterschiedlich sein. Nur wenn der Computer bei der Auswertung dieses Ausdruckes am Ende den Wert wahr berechnet hat, wird zu Zeile 2210 gesprungen. In jedem anderen Fall (irgendwelche zwei Ziffern sind gleich), erscheint die Meldung aus Zeile 2200.

Wie ich weiter oben erklärt habe, legt der Computer Ihre Variablen selbständig an, das heißt, sobald ein Variablenname zum ersten Mal im Programm erscheint, bemerkt das Ihr C64 und belegt die Variable mit Null. Jetzt werden Sie unter Umständen genervt fragen, was soll dann diese überflüssige Zeile 2210?

Dann möchte ich darauf hinweisen, daß der Computer die Variablen nur beim ersten Mal auf Null setzt. Und wenn Sie mehrere Versuche benötigen, um die Zahl zu erraten, trifft der C64 jedesmal auf diese Zeile. Wenn diese Zeile fehlte, hätten Sie beim zweiten Versuch schon völlig wirre Versuchsauswertungen. So einen ähnlichen Fall hatten wir schon einmal in Zeile 2100 (T=0).

Probieren Sie es einmal: Gehen Sie mit Ihrem Cursor auf die Zeile 2210, drücken Sie dreimal [SHIFT]+[INST/DEL] und geben Sie jetzt hinter der Zeilennummer »REM« ein ([RETURN] nicht vergessen!). Ist Ihnen bewußt, was Sie damit gemacht haben?

Sie haben diese Zeile soeben zu einem Kommentar umfunktioniert; sobald Ihr Programm an dieser Stelle angekommen ist, überspringt der Computer diese Zeile. »REM«-Einfügungen sind ein bequemes Mittel, um Programme zu testen oder zu verstehen.

Wenn Sie diese kleine Veränderung vorgenommen haben, wage ich zu behaupten, daß Sie das richtige Ergebnis nicht oder höchstens rein zufällig herausfinden. Es kann Ihnen nämlich passieren, daß bei der Auswertung auf einmal »OOOOO« dasteht, und das bei drei Zahlen.

Sie dürfen also niemals vergessen, Variablen mit einem Wert vorzubelegen, vor allem, wenn der Programmteil mehrmals durchlaufen wird.

Wenn Sie Ihr Programm wieder reparieren möchten, brauchen Sie bloß auf die Zeile 2210 zu fahren und den Befehl »REM« wieder zu entfernen.

Ich möchte nun noch die Variablennamen erklären:

SR Zahl richtig und an der richtigen Stelle
SF Zahl richtig, aber an der falschen Stelle
RE Restliche (falsche) Zahlen

In den Zeilen 2220 bis 2270 werden Ihre eingegebenen Ziffern mit den Zufallsziffern des Computers verglichen. Ich hoffe, die Abfragen bereiten jetzt keine Schwierigkeiten mehr. Zur Sicherheit werden wir sie aber noch einmal kurz ansprechen:

In den ersten drei Zeilen wird die Variable »SR« erhöht, falls die eingegebenen Ziffern in Wert und Stelle mit der Zufallszahl übereinstimmen. In den anderen drei Zeilen wird »SF« erhöht, falls eine Ihrer eingegebenen Ziffern mit den Ziffern der Zufallszahl, aber an einer anderen Stelle übereinstimmt.

Die Variable »RE« berechnet sich dann ganz leicht aus den übriggebliebenen Möglichkeiten (Zeile 2280).

In den Zeilen 2290 wird die Auswertung Ihrer Eingabe auf den Bildschirm gedruckt. Hier gibt es jetzt mal wieder einen neuen Befehl:

Die FOR-NEXT-Schleife. Ein Computer ist hauptsächlich dazu da, immer wiederkehrende Aufgaben zu erledigen. Wenn Sie in einem Programm immer wieder dasselbe ausführen lassen wollen, muß es nicht für jeden einzelnen Durchlauf neu programmiert werden. Dafür gibt es diese Schleife. Sie benötigen dazu lediglich eine Variable, wie in unserem Beispiel »A«. Dieser Variable weisen Sie eine untere und eine obere Grenze zu. Das geschieht mit der Anweisung »FOR A=1 TO 10«. In diesem Fall wurden der Variablen die Grenzen 1 und 10 zugeordnet. Jetzt kommt in Ihrem Programm der Teil, der mehrere Male abgearbeitet werden soll. Und nach diesem Teil steht die Anweisung »NEXT A«. Somit können Sie beliebig große Programmstrukturen wiederholen lassen. Die Schleife oben wird

insgesamt zehnmal durchlaufen werden. Sie können natürlich auch beliebige andere Zahlen benutzen. Beispiel: FOR XY=100 TO 300:PRINT "HALLO":NEXT XY. Diese Schleife wird Ihnen 201 mal den Text »HALLO« auf dem Bildschirm ausgeben.

Zurück jetzt zu unserem Programm: Die Variablen »SR«, »SF« und »RE« enthalten nach der Auswertung die Zahlen, wie viele Ziffern ganz, teilweise oder gar nicht richtig waren. Mit Hilfe dieser Variablen werden nun die entsprechenden Zeichen »*«, »O« und »=« ausgegeben.

Hat die Variable SR zusätzlich den Wert drei (alle drei Ziffern waren richtig), so läßt das Programm den Cursor noch um eine Zeile nach unten springen und kehrt dann in das Hauptprogramm zurück (Zeile 2340).

In Zeile 2350 wird die Variable »DG« um eins pro Runde erhöht; danach springt das Programm von Zeile 2370 in die Zeile 2100. Dort steht beim INPUT-Befehl hinter dem Anführungszeichen aber kein inverses Herz, sondern ein großes inverses S; dieses »S« steht nicht für [SHIFT]+[CLR/HOME], sondern für [CLR/HOME] allein; somit wird der Bildschirm nicht bei jeder Runde gelöscht. Und nur deshalb hat die Zeile 2120 eine Daseinsberechtigung, da sie eine Zeile auf dem Bildschirm löschen soll; ansonsten würden sich die Bildschirmausgaben bald überschneiden, und Sie könnten nichts mehr erkennen. Sie merken also, daß kleine Ursachen (zum Beispiel Weglassen der [SHIFT]-Taste) große Auswirkungen haben können.

Von Zeile 90 aus geht es in den Endteil. Dort werden Sie jetzt keine neuen Befehle mehr finden. Ich verlasse mich auf Sie. Ich hoffe, Sie werden mir keine Schande machen. Es wäre das Beste, wenn Sie noch den restlichen Programmteil analysieren. Eigentlich dürfte Ihnen das keine Schwierigkeiten bereiten, da nur relativ einfache Befehle vorkommen.

Auf eine mögliche Fehlerquelle möchte ich noch hinweisen: In Zeile 3050 wird die Meldung ausgegeben, wenn Sie weniger als sechs Durchgänge benötigten. In der nächsten Zeile erfolgt eine Ausgabe, wenn Sie weniger als elf Runden benötigten, in Zeile 3070 für alle anderen Fälle. Sollten Sie aber an den Enden der Zeilen 3050 und 3060 jeweils den GOTO-Befehl vergessen, würde das zum Beispiel für den Fall DG=5 bedeuten, daß alle drei Ausgaben erfolgen; schließlich sind bei beiden IF-Abfragen die Bedingungen erfüllt.

Abschließend muß ich zugeben, daß das für den Anfang ein dicker Brocken war. Aber dadurch, daß jeder einzelne Befehl erklärt werden mußte, wurde die Erklärung zu diesem Spiel auch sehr lang. Bereits der letzte Teil bedarf eigentlich keiner Erklärung mehr, da Sie alle vorkommenden Befehle kennen. In Zukunft können die Anleitungen und Erklärungen also etwas kürzer gehalten werden.

2 König

Einführung

Für den C64 gibt es ein professionelles Spiel namens Kaiser. In diesem Spiel müssen Sie sich vom Landadel bis zum Kaiser allmählich hocharbeiten. Dabei werden Ihre Verdienste um Ihr Volk, Gerechtigkeit etc. bewertet.

Diesem Programm wurde König nachempfunden. Allerdings wurde das Programm stark vereinfacht (man beachte den Namen). In unserer Version können Sie lediglich Land kaufen und die Menge des Getreides, das an die Bevölkerung ausgeteilt werden soll, beeinflussen. Die restlichen Größen wie Einwohner, Einwanderer, Verstorbene, Landpreis, Getreide sowie Getreidepreis werden mit dem Zufallsgenerator erstellt.

Mit diesem Grundgerüst sollen Sie nun etwas anfangen! Erweitern Sie das Programm, lassen Sie Ihrer Fantasie freien Lauf. Sollten Sie nicht wissen, was Sie tun könnten, möchte ich ein paar Anregungen geben:

Lassen Sie auch Landverkauf zu. Verändern Sie das Programm so, daß die Anzahl der Toten von der Menge des ausgeteilten Getreides *und* der Größe der Bevölkerung abhängt. Führen Sie eine Bewertung für die Taten des Königs ein.

Falls Sie eine solche Bewertung eingeführt haben, könnten Sie auch die Zahl der Einwanderer von der Bewertung im letzten Jahr abhängig machen. Sollten Sie vorhaben, das Programm stark zu erweitern, kann ich nur raten, das Listing in diesem Buch nur als Grundlage zu sehen, und praktisch ein völlig neues Spiel zu programmieren. Ansonsten müßten Sie Programmzeilen einfügen, alles Bisherige verändern...

In einem solchen Fall ist es immer besser, ein völlig neues Programm aufzubauen. Das erleichtert Ihnen die Arbeit ungemein. Natürlich können Sie in Ihrer eigenen Version Teile des Spiels übernehmen. Aber es ist doch ein viel besseres Gefühl, wenn man ein Programm ganz alleine schreibt.

Ich wünsche Ihnen viel Erfolg bei Ihren ersten Programmierschritten. Sollten Sie sich dieses Projekt noch nicht zutrauen, habe ich volles Verständnis dafür. Aber wenn Sie in diesem Buch ein bißchen weiter sind, sollten Sie sich an diese Aufgabe wagen.

Listing zu König

```
10 REM **************
20 REM ***        ***
30 REM *** KOENIG ***
40 REM ***        ***
50 REM **************
60 REM
70 GOSUB 1000
80 GOSUB 2000
90 GOSUB 3000
100 END
1000 REM
1010 REM *** VORBEREITUNGEN ***
1020 REM
1030 E=100
1040 EI=5
1050 T=0
1060 GG=4000:RG=300
1070 LG=GG-RG
1080 J=0
1090 L=1000:DIM JZ$(15)
1100 FOR A=1 TO 15:READ JZ$(A):NEXT A
1110 DATA ERSTEN, ZWEITEN, DRITTEN, VIERTEN, FUENFTEN
1120 DATA SECHSTEN, SIEBTEN, ACHTEN, NEUNTEN, ZEHNTEN, ELFTEN
1130 DATA ZWOELFTEN, DREIZEHNTEN, VIERZEHNTEN, FUENFZEHNTEN
1140 RETURN
2000 REM
2010 REM *** SPIEL ***
2020 REM
2030 J=J+1
2040 PRINT "{CLR,DOWN}";TAB(15);"{RVSON,SPACE}KOENIG{SPACE,RVOFF}"
2050 PRINT "{2DOWN}JAHRESBERICHT FUER DEN KOENIG:"
2060 PRINT "{DOWN}WIR BEFINDEN UNS IM ";JZ$(J);" JAHR."
2070 PRINT "{2DOWN}IN DIESEM JAHR WURDEN:{DOWN}"
2080 PRINT E;TAB(6);"EINWOHNER REGISTRIERT"
2090 PRINT EI;TAB(6);"EINWANDERER AUFGENOMMEN"
2100 PRINT T;TAB(6);"PERSONEN BEERDIGT"
2110 PRINT TAB(6);"{DOWN}VON DEINEN";L;"KM↑2 LAND{DOWN}"
2120 PRINT GG;TAB(6);"KG GETREIDE GEERNTET;"
2130 PRINT RG;TAB(6);"HABEN DIE RATTEN GEFRESSEN."
2140 PRINT LG;TAB(6);"KG SIND NOCH UEBRIG."
2150 PRINT TAB(6);"{2DOWN}BITTE EINE TASTE DRUECKEN !"
2160 GET A$:IF A$="" THEN GOTO 2160
2170 IF J=15 THEN RETURN
2180 LP=INT (RND(1)*27)+1
2190 PRINT "{CLR,3DOWN}LANDPREIS:";LP;"KG GETREIDE"
2200 PRINT "GETREIDEVORRAT:";LG;"KG"
2210 PRINT "{2DOWN}WIE VIELE KM↑2 LAND MOECHTEN SIE DAZU-"
2220 PRINT "KAUFEN ? DERZEITIGER BESITZ:";L;"KM↑2"
2230 AN=0:INPUT "IN KM↑2 ";AN
2240 IF AN*LP<LG THEN GOTO 2260
2250 PRINT "SCHULDEN DUERFEN SIE KEINE MACHEN !":GOTO 2230
2260 IF AN>0 THEN L=L+AN:LG=LG-AN*LP
2270 PRINT "{DOWN}NEUER GETREIDEVORRAT:";LG;"KG"
2280 PRINT "{DOWN}WIEVIEL DAVON SOLL AN DIE";E
2290 PRINT "EINWOHNER VERTEILT WERDEN ?"
2300 AN=0:INPUT "IN KG ";AN
2310 IF AN>LG THEN PRINT "SOVIEL HABEN SIE GAR NICHT !":GOTO 2300
2320 IF AN<10 THEN PRINT "DIE MENSCHEN WERDEN VERHUNGERN !":GOTO 2300
2330 IF LG-AN<2 THEN PRINT "BEHALTEN SIE EINEN{2SPACE}KLEINEN VORRAT !'
     GOTO 2300
2340 LG=LG-AN
2350 PRINT "{DOWN}NEUER GETREIDEVORRAT:";LG;"KG"
```

```
2360 PRINT "{2DOWN}BITTE EINE TASTE DRUECKEN ..."
2370 GET A$:IF A$="" THEN GOTO 2370
2380 T=INT (RND (1)*(0.5*E))
2390 IF T<0.4*E THEN GOTO 2510
2400 IF E=0 THEN RETURN
2410 PRINT "{CLR,3DOWN}TJA, DAS TUT MIR WIRKLICH SEHR LEID,"
2420 PRINT "ABER SIE SIND DAS OPFER EINES AUFSTANDES";
2430 PRINT "GEWORDEN. DIE AUFGEBRACHTEN MASSEN HABEN";
2440 PRINT "SIE AUS DEM LAND GEJAGT."
2450 PRINT "DIE STERBERATE WAR MIT";INT (T/E*1000)/10;"%"
2460 PRINT "ETWAS ZU HOCH."
2470 PRINT "{DOWN}DABEI WAR ES DEM VOLK WIRKLICH EGAL, OB"
2480 PRINT "SIE NUN SCHULD DARAN WAREN, ODER NICHT."
2490 PRINT "MAN BRAUCHTE EBEN EINEN SUENDENBOCK."
2500 GS=1:RETURN
2510 E = INT (RND (1)*(0.9*E)) + E + EI
2520 EI= INT (RND (1)*9)
2530 GG= INT (RND (1)*5*L)
2540 RG= INT (RND (1)*(0.6*GG))
2550 LG=GG-RG+LG
2560 GOTO 2030
3000 REM
3010 REM *** ENDE ***
3020 REM
3030 IF GS=1 THEN RETURN
3040 PRINT "{CLR}NACH FUENFZEHN JAHREN IHRER HERRSCHAFT"
3050 PRINT "BEVOELKERN";E;"IHR LAND."
3060 PRINT "ANGEFANGEN HABEN SIE MIT 100 MENSCHEN."
3070 PRINT "DAS ERGIBT EINE DIFFERENZ VON";E-100
3080 IF 100-E>0 THEN GOTO 3110
3090 PRINT "{DOWN}PRO JAHR WAR DAS EIN DURCHSCHNITTLICHER"
3100 PRINT "ZUWACHS VON";(E-100)/15*100:GOTO 3130
3110 PRINT "PRO JAHR WAR DAS EINE DURCHSCHNITTLICHE"
3120 PRINT "ABNAHME VON";(E-100)/15*100
3130 PRINT "{2DOWN}IHR LANDBESITZ VERGROESSERTE SICH VON"
3140 PRINT "1000 KM↑2 AUF";L;"KM↑2"
3150 PRINT "{2DOWN}ICH HOFFE, DAS SPIEL HAT IHNEN SPASS"
3160 PRINT "GEMACHT. BEEHREN SIE UNS MIT IHRER"
3170 PRINT "REGIERUNG BALD WIEDER."
3180 RETURN
```

■ Programmbeschreibung für König

Nachdem Sie sich durch Superhirn mehr oder weniger durchgequält haben, können Sie sich bei König jetzt ein wenig entspannen. Bei diesem Programm ist der größte Teil nur für die Bildschirmausgabe zuständig, so daß Sie also ein gutes Beispiel dafür sehen, was man alles mit dem PRINT-Befehl anstellen kann.

Das Gute an diesem Programm ist, daß man es fast beliebig erweitern kann. Daher stellt dieses Spiel auch nur ein grobes Gerüst für eigene Ideen dar. Doch dazu am Ende mehr. Zunächst versuchen wir, diesem Programm auf den Grund zu gehen.

Mit meinen Erklärungen beginne ich aber erst ab Zeile 1030. Die Befehle davor sollten Sie im Schlaf beherrschen. Zunächst beginnt das Programm mit den Initialisierungen (daher auch der Name des Unterprogramms: Vorbereitungen). Allen Variablen wird ein

bestimmter Wert zugewiesen. Schließlich müssen Variablen bei Spielbeginn nicht immer den Wert 0 enthalten.

In den Variablen wird die Ausgangssituation für Ihr Land festgelegt. Dabei haben diese Werte nur für die erste Runde ihre Gültigkeit (wie wir später noch sehen werden).

Für die erste Runde wird aber festgelegt, daß es in Ihrem Land

100	Einwohner,
5	Einwanderer und
0	Tote gibt. Außerdem werden von Ihren Feldern
4000 kg	Getreide geerntet. Allerdings fressen die Ratten
300 kg	davon, so daß in Ihren Lagern
3700 kg	Vorrat liegen.

Wir befinden uns im ersten Jahr Ihrer Herrschaft. Die Anzahl dieser Jahre wird in der Variablen »J« festgehalten. Da diese Variable aber immer zu Beginn einer neuen Runde um eins erhöht wird (siehe Zeile 2030), muß sie vor dem ersten Jahr auf 0 stehen. Man könnte sich diese Anweisung also sparen; sie wurde aber der Vollständigkeit halber aufgeführt.

In der Variablen »L« wird Ihr Landbesitz festgehalten. Er beträgt zu Beginn des Spiels 1000 Quadratkilometer. In Zeile 1090 sehen Sie nach der eben erwähnten Variablen einen neuen Befehl, den DIM-Befehl. Um diesen Befehl näher erläutern zu können, muß ich etwas weiter ausholen:

Bisher haben Sie Variablen kennengelernt, die (ganze) Zahlen aufnehmen können. In der Theorie wissen Sie aber schon, daß es Variablen gibt, die auch Zeichenketten (Strings genannt) verarbeiten. Was jetzt kommt, ist kein neuer Variablentyp, sondern eine Zusammensetzung von Variablen. Wenn Sie in einer Variablen zur gleichen Zeit mehrere Werte speichern wollen, müssen Sie mit dieser Methode arbeiten.

Ich möchte Ihnen das an einem praktischen Beispiel verdeutlichen. Sie wollen eine Liste aller Spiele aus diesem Buch, die Sie bereits durchgearbeitet haben, mit dem C64 verwalten. Eine Möglichkeit wäre natürlich, so viele Variablen einzuführen, wie Spiele vorhanden sind. Zum einen wird das extrem viel Tipparbeit, und zum anderen ist Ihr Programm in keinster Weise flexibel, sollten mehr Programme dazukommen, als ursprünglich angenommen. Jetzt kann man auf einen Trick zurückgreifen:

Sie benutzen eine Feldvariable. Das heißt, daß Sie eine Variable benutzen, diese Variable aber indizieren, und somit aus einer Variablen beliebig viele machen. In Basic sieht das Ganze wie folgt aus: Anstelle vieler einzelner Variablen (zum Beispiel A, B, C, D, E, F, G, H, I, J ...) benützen Sie eine einzige Variable (zum Beispiel A(I)). Dieses »I« kann beliebige ganze Werte über 0 annehmen (0, 1, 2, 3...). Natürlich muß dieser Index nicht »I« heißen; da »I« aber oft als Zählvariable benutzt wird, konnte es auch in diesem Fall herhalten.

Sie können also mittels dieses Index auf alle Teile der Variable »A« zugreifen. Bevor Sie jedoch eine solche Feldvariable benutzen, müssen Sie dies dem C64 mitteilen, und zwar mit dem Befehl DIM. Nach »DIM« kommt der Name der Variablen (in unserem Fall »A«) und in Klammern dahinter die Zahl der Elemente.

Beispiel: Sie wollen in der Feldvariablen »G« Ihr Einkommen der letzten zwölf Monate speichern. In einem Programm könnte das so aussehen:

```
10 DIM G(12)
20 FOR A=1 TO 12
30 :INPUT "IHR GEHALT ";G(A)
40 NEXT A
50 FOR A=1 TO 12
60 :PRINT "IHR GEHALT";G(A)
70 NEXT A
```

Natürlich ist das ein sehr einfaches Beispiel, aber ich hoffe, die Benutzung von Feldvariablen wurde dadurch klar. Stellen Sie sich vor, Sie hätten zwölf Variablen eingeführt. Sie müßten dann für jede Variable einen eigenen INPUT-Befehl und einen eigenen PRINT-Befehl verwenden. Außerdem können Sie dieses Programm beliebig auf 24, 36 oder noch mehr Gehälter erhöhen, indem Sie einfach die 12 in den Zeilen 10, 20 und 50 verändern. Bei der anderen Version müßten Sie pro weiteren Monat zwei Zeilen anhängen...

Das war jetzt ein weiter Exkurs, nur um den kleinen DIM-Befehl zu erklären. Aber eine Anmerkung noch: Sie werden selten den Ausdruck Feldvariable finden. Meistens werden solche Felder Arrays genannt.

Nicht genug damit, daß Sie mit dem neuen Befehl in Zeile 1090 konfrontiert werden, da sehen Sie auch noch zum ersten Mal den Variablentyp String, der Zeichenketten verwaltet. Wie bereits bekannt ist (oder sein sollte?), erkennt der C64 einen String an dem nachgestellten $-Zeichen. Ansonsten können Sie mit diesen Variablen genauso umgehen wie mit den Zahlenvariablen.

In der Zeile 1100 wartet schon wieder ein neuer Befehl auf Sie, der READ-Befehl. Wenn Sie in einem Programm Daten speichern wollen, die bei jedem Programmlauf eingelesen werden sollen, die sich aber nicht verändern, so können Sie diese Daten in einem Programm mit dem DATA-Befehl speichern (siehe Zeilen 1110 bis 1130). Diese Daten werden durch Kommata getrennt und mit dem READ-Befehl eingelesen.

Wenn wir noch einmal das Beispiel von vorher verwenden und davon ausgehen, daß Ihr Gehalt bei jedem Programmlauf eingelesen werden soll, so könnte das folgendermaßen aussehen:

```
10 FOR A=1 TO 12
20 :READ G(A)
30 NEXT A
40 DATA 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000
50 DATA 9000, 10000, 11000, 12000
```

In diesem Beispiel gehen wir davon aus, daß Ihr Einkommen jedes Jahr denselben Schwankungen unterworfen ist. Ich hoffe, Sinn und Zweck dieser beiden Befehle wurde jetzt verständlich. Allerdings gibt es da noch ein Problem: Wenn man mit READ Daten

ausliest, wird im Computer ein Zeiger verwaltet, der immer auf die nächsten Daten zeigt. Nur so ist es ja möglich, mehrere Daten zu speichern. Sobald dieser Zeiger aber beim letzten DATA-Eintrag angekommen ist, und Sie immer noch mehr Daten mit dem READ-Befehl anfordern, wird der Computer eine Fehlermeldung (`?OUT OF DATA ERROR`) ausgeben. Jetzt gibt es zwei Möglichkeiten. Entweder lassen Sie den Zeiger wieder auf den allerersten Eintrag zeigen, oder Sie hängen weitere Daten an.

Für die erste Methode gibt es einen Befehl: RESTORE. Er richtet den Zeiger wieder so ein, daß er auf das erste Element (in unserem kleinen Beispiel auf 1000) zeigt.

Das müssen Sie beachten, zum Beispiel auch, wenn Sie die Liste ein zweites Mal einlesen lassen wollen. In den Zeilen 1110 bis 1130 sehen Sie jetzt die Daten, die mit dem READ-Befehl dem Array »JZ$« (wie JahresZahl) zugeordnet werden. Man kann gut erkennen, wie die einzelnen Daten durch Kommata getrennt sind. Beachten Sie aber bitte, daß Sie am Ende einer Zeile kein Komma setzen dürfen, auch wenn in der nächsten Zeile weitere Daten folgen.

Mit »RETURN« in Zeile 1140 wird bekanntermaßen das Unterprogramm abgeschlossen, und der Interpreter kehrt zu Zeile 70 bzw. Zeile 80, da in 70 keine weiteren Befehle mehr stehen, zurück.

Der erste Befehl im Programmteil »SPIEL« erhöht die Jahreszahl um eins. Diese Variable wurde eingeführt, damit das Programm nach 15 Jahren das Spiel selbständig beendet. Dadurch soll eine Diktatur Ihrerseits ausgeschlossen werden. Allerdings ist es fraglich, ob sich jemals ein Herrscher solang auf dem Thron wird halten können. Ihr Land neigt nämlich zu Revolutionen ...

Anhand der folgenden Zeilen können Sie erkennen, wie man mit dem PRINT-Befehl sinnvoll umgehen kann. In Zeile 2040 wird zuerst der Bildschirm gelöscht (mit [SHIFT]+[CLR/HOME]), dann springt der Cursor eine Zeile tiefer (mit [CRSR-DOWN]). Danach zeigt dem C64 das zweite Anführungszeichen, daß die direkte Bildschirmausgabe jetzt beendet ist; der Strichpunkt bedeutet, daß die nächsten Zeichen direkt an der Stelle gedruckt werden, an der der Cursor im Moment steht. Danach kommt eine Anweisung, mit der man jede einzelne Spalte des Bildschirms anspringen kann (siehe auch Beschreibung von Superhirn). In dem Fall wird der Computer also veranlaßt, die Schreibmarke (Cursor) an Spalte 15 zu setzen. Nach dem Strichpunkt geht es innerhalb der Anführungszeichen weiter mit der direkten Bildschirmausgabe. Dabei sehen Sie ein neues Zeichen, dieses inverse »R« (invers bedeutet helles Zeichen auf dunklem Grund, also genau andersherum als gewöhnlich). Innerhalb der Anführungsstriche erreichen Sie dieses Zeichen, indem Sie [CTRL]+[9] drücken. Wenn Sie auf die Vorderseite der Taste [9] schauen, erkennen Sie, was diese Tastenkombination bewirkt. Dadurch wird eben der Invers-Modus eingeschaltet. Ausschalten kann man ihn wieder mit [CTRL]+[0]. Dieses Umschalten funktioniert übrigens genauso im Direktmodus, das heißt ohne Anführungszeichen.

In Zeile 2060 wird in die direkte Ausgabe eine Variable eingeschoben. Das veranlaßt den Computer, den Inhalt dieser Variablen auszugeben. In unserem Fall enthält »J« die Jahreszahl und »JZ$(J)« den entsprechenden Text zur jeweiligen Jahreszahl.

Die folgenden Zeilen bis 2150 funktionieren alle nach demselben Muster. Jedesmal wird eine Variable ausgegeben, der Cursor in die Spalte 6 gesetzt und dann noch ein bestimmter Text ausgegeben. Allmählich sollten Sie mit dem PRINT-Befehl vertraut werden.

In Zeile 2160 sehen Sie wieder den GET-Befehl. Beachten Sie bitte immer, daß eine sogenannte Leereingabe mit einer IF-Abfrage abgefangen werden muß. Bei diesem Beispiel ist es sinnvoll, die IF-Abfrage noch in dieselbe Zeile zu setzen wie den GET-Befehl selbst, da beide Ausdrücke direkt voneinander abhängen. Es wäre auch nicht sinnlos, beiden Befehlen eine eigene Zeile zu widmen, dadurch gewinnt man aber keinerlei Übersichtlichkeit.

In Zeile 2170 steht die oben bereits angesprochene Abfrage, die verhindern soll, daß Sie eine Diktatur aufbauen können, sprich, die keine längere Regierungszeit als 15 Jahre zuläßt. Erinnern Sie sich noch an die Zufallsfunktion, sowie die INTeger-Funktion? Beide tauchen in der nächsten Zeile wieder auf. Demnach sollte es Ihnen keine Schwierigkeiten bereiten zu erkennen, daß die Variable »LP« (für LandPreis) Werte zwischen 1 und 27 annehmen kann.

In den nächsten vier Zeilen kommen wieder nur Bildschirmausgaben vor, deren Erläuterung ich mir spare. Erst Zeile 2230 wird wieder interessant, auch wenn nichts Neues geboten wird. Wie Sie bereits wissen, können Sie mit dem INPUT-Befehl Eingaben des Benutzers verarbeiten. Da auch hier die ANtwortvariable »AN« mit 0 vorbelegt worden ist, können Sie beruhigt [RETURN] drücken, wenn Sie kein Land kaufen wollen. Sie müssen also nicht jedesmal noch die [0] drücken. Bei Ihren Landkäufen sollten Sie beachten, daß die Kosten nicht Ihr Vermögen übersteigen. Das Programm überprüft das zwar in Zeile 2240 und 2250, aber die Eingabe dauert ja nur um so länger.

Die IF-Abfrage dürfte eigentlich nicht zu schwierig sein: In »AN« ist gespeichert, wie viele Quadratkilometer Land Sie kaufen wollen, in »LP« ist der Landpreis pro Quadratkilometer gespeichert, während »LG« Ihre Lagervorräte enthält. Sie sollten beachten, daß Sie einen kleinen Getreidevorrat behalten, den Sie dann an das Volk verteilen können.

Bisher ist in dem Programm nur Landkauf vorgesehen. Wenn Sie glauben, das Programm mit negativen Angaben überlisten zu können, haben Sie sich getäuscht; dafür sorgt die Abfrage in Zeile 2260, die die Abrechnung nur vornimmt, wenn Ihre Eingabe positiv war. Aber sobald Sie diese Abfrage entfernen, müßten Sie auch Land verkaufen können. Besser wäre es natürlich, wenn Sie das Programm um einen eigenen Eingabeteil für den Verkauf von Land erweiterten.

Wenn Sie sich die Zeilen 2270 bis 2370 anschauen, so werden Sie sehen, daß alle Befehle analog zu oben arbeiten, nur diesmal nicht für Ihre Ländereien, sondern für den Getreidevorrat.

Ab Zeile 2380 werden die Vorbereitungen für die nächste Runde getroffen: Per Zufallsgenerator wird die Zahl der Verstorbenen gesteuert; sie schwankt zwischen 0 und maximal 50% der Bevölkerung. Sollte »T« kleiner als 40% der Bevölkerung sein, so verzweigt das Programm zu Zeile 2510, um die weiteren Vorbereitungen zu treffen. Andernfalls werden Vorkehrungen für Ihren Sturz getroffen...

Sollte die Zahl der Einwohner mittlerweile auf 0 gesunken sein, wird das Spiel sofort beendet. Andernfalls, das heißt, die Zahl der Toten liegt zwischen 40% und 50% der Gesamtbevölkerung, und es gibt noch Einwohner, erhalten Sie noch ein paar Informationen, warum Sie gehen mußten. Das einzige, was an diesen paar Zeilen (2410–2500) interessant sein dürfte, ist Zeile 2450. Mit der Division »T/E« wird der Anteil der Toten an der Gesamtbevölkerung gemessen; dieses Ergebnis wird mit 1000 multipliziert, die Stellen hinter dem Komma werden mittels INT abgeschnitten, und dieses Ergebnis durch zehn dividiert. »T/E« müßte mit 100 multipliziert werden, um ein Ergebnis in Prozent zu erhalten; dadurch aber, daß man hier zuerst mit 1000 multipliziert und anschließend wieder durch zehn dividiert, erhält man die Prozentangaben mit einer Stelle hinter dem Komma. Sobald Sie das erste Mal gestürzt werden, werden Sie dieses Ergebnis betrachten können.

Zum Abschluß Ihres Sturzes wird noch die Variable »GS« (wie GeStürzt) auf 1 gesetzt. Sie werden später noch sehen, wofür diese Variable vorgesehen ist.

Wie gesagt werden ab Zeile 2510 die Vorbereitungen für eine neue Runde getroffen. Die Zahl der Toten wurde bereits berechnet; die Zahl der Einwohner (E) berechnet sich aus Einwohner aus diesem Jahr plus Einwanderer von diesem Jahr plus maximal 90% der Bevölkerung. Das heißt also, daß die Bevölkerung aus eigenen Kräften nicht mehr als um 90% (90/100=0.9) steigen kann. Sie meinen, die Toten würden gar nicht berücksichtigt? Stimmt schon, aber das soll eine Anregung für Sie sein. Wenn Sie das Programm »korrigieren« wollen, ist einiges zu beachten: In Zeile 2380 wird bereits die Zahl der Toten für das nächste Jahr bestimmt. Das heißt, die Berechnung für die neue Einwohnerzahl müßte vor 2380 geschehen (wenn Sie die Zwischenspeicherung der Totenzahl vermeiden wollen). Sie können aber die Zeile 2510 nicht ohne weiteres vorverlegen, da sonst der GOTO-Befehl in 2390 nicht die Zeile findet, die er anspringen soll; das heißt, der Interpreter bricht mit einer Fehlermeldung ab ...

Die Zahl der Einwanderer schwankt zwischen 0 und 8, während das geerntete Getreide (GG) maximal das Fünffache Ihrer Ländereien betragen kann. Das soll natürlich nicht heißen, daß Ihr Getreide auf einmal in Quadratkilometern gemessen wird, sondern lediglich bedeuten, daß die Ernte von der Feldgröße abhängt.

Die Ratten erhalten maximal 60% des geernteten Getreides, und in Ihren Lagerhallen türmt sich jetzt das Getreide von diesem Jahr (LG) plus das geerntete Getreide (GG) minus dem Rattenanteil (RG). Nachdem diese Vorarbeit geleistet wurde, kehrt das Programm zu Zeile 2030 und somit einer neuen Runde zurück.

Das Ende möchte ich jetzt nur noch kurz ansprechen. Sollten Sie gestürzt worden sein (GS=1), dann erhalten Sie in diesem Teil keine Meldungen mehr; Ihnen wurden die Gründe für Ihren Sturz schon in den Zeilen 2410ff. ausführlich genug auseinandergesetzt.

In den anderen Fällen wurde zwar keine Variable GS auf 0 gesetzt, aber Sie erinnern sich: Sobald eine Variable zum ersten Mal auftaucht, wird sie vom C64 mit 0 vorbelegt. Sollten Sie nicht gestürzt worden sein, sondern nach 15 Jahren Herrschaft abdanken müssen, so taucht für den Interpreter die Variable »GS« in Zeile 3030 zum ersten Mal auf; er ist ja noch nicht in Zeile 2500 gekommen, also weiß er auch nicht, daß dort diese Variable schon einmal vorkommt. Folglich ist für ihn GS=0, das heißt die Bedingung in 3030 nicht erfüllt, und das wiederum bedeutet, daß jetzt alle Zeilen danach bearbeitet werden. Und was da Ihr armer C64 zu tun hat, das überlasse ich jetzt Ihnen: Versuchen Sie, diesen letzten Teil selbständig zu analysieren und zu verstehen. Es sollte wirklich nicht allzuschwer sein. Überhaupt sollte die Erklärung für dieses Spiel eigentlich nicht so lang werden ...

Ich habe zwar schon eine Anregung mitgegeben, aber ich finde, dieses Spiel eignet sich gut für Weiterentwicklungen. Man kann es fast beliebig erweitern und neue Spielteile dazuerfinden. Mir ist klar, daß Sie jetzt zu Beginn noch nicht so sicher sind, als daß Sie alle gestellten Probleme auf Anhieb selbständig lösen könnten. Aber vielleicht nehmen Sie sich diese Erweiterungen für später vor?

Eine Bewertung des Spielers würde dem Spiel sicher einen neuen Spielreiz verleihen. Stellen Sie sich vor, die Taten des Königs würden bewertet werden; wollte man das Programm noch stärker erweitern, könnte man den Spieler als Graf anfangen lassen und je nach Bewertung langsamer oder schneller aufsteigen lassen.

Neue Abfragen wären aber das Mindeste, was eingebaut werden sollte. So dürfte es nicht möglich sein, 1000 Einwohnern nur 10 kg Getreide zur Verfügung zu stellen. Daher sollte das Programm überprüfen, ob jedem Einwohner genügend Land und genügend Getreide zur Verfügung gestellt werden, da die Leute sonst wegen Überbevölkerung bzw. Hungersnot sterben. Das ließe sich unter Umständen wieder in die Bewertung des Herrschers einbauen.

Wie wäre es auch mit einer Spielfassung für zwei Spieler?

Viel Spaß und viel Erfolg beim Programmieren!

3 Schlammschlacht

Einführung

Dieses Spiel ist ganz schnell erklärt. Sie müssen mit Ihren Würfen den Drachen Grisi treffen. Sie spielen nämlich mit ihm in einer Schlammpfütze, und Grisi liebt es nun einmal über alles, von Ihnen »eingeseift« zu werden (eine Namensähnlichkeit mit Feuerwehrdrachen wäre übrigens rein zufällig).

Das Programm zeigt die Koordinaten von Grisi an; danach müssen Sie sich richten, wenn Sie den Winkel und die Höhe für Ihren Wurf eingeben. Für diese Eingaben möchte ich noch ein paar Tips geben:

Wie nicht anders zu erwarten, fliegt ein Wurf bei 45 Grad am weitesten.

Für die X/Y-Koordinaten und den Wurfwinkel gebe ich noch eine kleine Tabelle mit auf den Weg, die Ihnen das Leben etwas erleichtern soll:

Wurfwinkel	X-Koordinate	Y-Koordinate
0	1700	0
45	1400	1400
90	0	1700
135	-1400	1400
180	-1700	0
225	-1400	-1400
270	0	-1700
315	1400	-1400
360	1700	0

Den Rest kann ich nur Ihrer Experimentierlust überlassen; ganz nach dem Motto: Probieren geht über Studieren.

Listing zu Schlammschlacht

```
10 REM ***********************
20 REM ***                 ***
30 REM *** SCHLAMMSCHLACHT ***
40 REM ***                 ***
50 REM ***********************
60 REM
70 GOSUB 1000
80 GOSUB 2000
90 GOSUB 3000
100 END
1000 REM
1010 REM *** VORBEREITUNGEN ***
1020 REM
1030 PRINT "{CLR}"
1040 RX=INT (RND (1)*2)+1
1050 RY=INT (RND (1)*2)+1
1060 X =INT (RND (1)*3000)
1070 Y =INT (RND (1)*3000)
1080 IF RX=1 THEN X=-X
1090 IF RY=1 THEN Y=-Y
1100 UR=3.1415926536/180
1110 RETURN
2000 REM
2010 REM *** SPIEL ***
2020 REM
2030 PRINT "{CLR,3DOWN}";TAB(8);"*** SCHLAMMSCHLACHT ***":DG=DG+1
2040 PRINT "{2DOWN}DER DRACHE GRISI HAT DIE KOORDINATEN:"
2050 PRINT "X-RICHTUNG:";X
2060 PRINT "Y-RICHTUNG:";Y
2070 PRINT "{DOWN}WELCHE HOEHE FUER DEN WURF ?"
2080 INPUT "1-90 ";WH
2090 IF WH<1 OR WH>90 THEN GOTO 2080
2100 PRINT "{DOWN}IN WELCHEM WINKEL SOLL ABGEWORFEN"
2110 INPUT "WERDEN ? (0-360) ";WW
2120 IF WW<0 OR WW>360 THEN GOTO 2110
2130 AB=ABS (INT (9000*SIN (WH*UR)*COS (WH*UR)))
2140 XW=INT (AB*SIN (WW*UR))
2150 YW=INT (AB*COS (WW*UR))
2160 AS=INT (SQR ((X-XW)↑2+(Y-YW)↑2))
2170 PRINT "{2DOWN}IHR SCHLAMMBALL TRAF FOLGENDE KOORDI-"
2180 PRINT "NATEN:"
2190 PRINT "X-RICHTUNG:";XW
2200 PRINT "Y-RICHTUNG:";YW
2210 PRINT "DER ABSTAND ZUM DRACHEN BETRAEGT DEMNACH"
2220 PRINT AS;"METER."
2230 GET A$:IF A$="" THEN GOTO 2230
2240 IF AS<1000 THEN RETURN
2250 GOTO 2030
3000 REM
3010 REM *** ENDE ***
3020 REM
3030 PRINT "{CLR,2DOWN}SIE BENOETIGTEN";DG;"VERSUCHE, UM GRISI"
3040 PRINT "ZU TREFFEN."
3050 IF DG<10 THEN GOTO 3080
3060 PRINT "GRISI IST ETWAS ENTTAEUSCHT VON IHREN"
3070 PRINT "WURFKUENSTEN. DENNOCH GIBT ER IHNEN NOCHEINE WEITERE CHANCE."
3080 PRINT "WOLLEN SIE GRISI NOCH EINMAL DEN GEFAL-"
3090 PRINT "LEN ERWEISEN UND IHN 'EINSEIFEN'?"
3100 GET AN$:IF AN$<>"J" AND AN$<>"N" THEN GOTO 3100
3110 IF AN$="J" THEN RUN
3120 RETURN
```

Programmbeschreibung für Schlammschlacht

Eigentlich bräuchte ich an diesem Programm (fast) gar nichts mehr zu erklären. Sie kennen praktisch alle Befehle, und da das Programm auch nicht allzu kompliziert ist, sollten Sie wenig Probleme haben, den Aufbau dieses Spiels zu erkennen. Aber keine Sorge, ich bin mir der Tatsache bewußt, daß Sie bislang noch keine so große Erfahrung im Umgang mit diesen Basic-Programmen haben, und ich werde Sie auch bestimmt nicht im Regen stehen lassen. Also:

Bis zur Zeile 1030 inklusive sollte Ihnen aber wirklich alles klar sein. In den folgenden vier Zeilen finden wir eine alte Bekannte wieder, die Zufallsfunktion. Sie ist genauso aufgebaut wie im Spiel Superhirn. Nur werden diesmal die Werte den Variablen »RX«, »RY«, »X« und »Y« zugeordnet. An dieser Stelle möchte ich gleich auf etwas hinweisen: Eine Sache ist es, ein Basic-Programm zu analysieren, eine andere ist es (noch zusätzlich), hinter die Bedeutung der Variablen zu kommen. Das ist in den seltensten Fällen einfach. In unserem Beispiel springt zwar der Sinn der Variablen nicht sofort ins Auge, aber man könnte schon nach kurzer Zeit dahinter kommen. Und zwar werden in den Variablen »X« und »Y« die Koordinaten des Drachens gespeichert. Mit RX und RY wird noch die Richtung bestimmt, in die Sie schauen müssen, um den Drachen zu »sehen«. Und zwar wird das mit Vorzeichen realisiert. Ist die Zufallsvariable RX=1, dann ist das Vorzeichen von »X« negativ; das Gleiche gilt analog für »RY« und »Y«. Stellen Sie sich ein Koordinatensystem vor, dann sollte Ihnen das System mit den Vorzeichen verständlich sein: Je nach Vorzeichen von »X« und »Y« (bestimmt durch »RX« bzw. »RY«) befindet sich der Drache im ersten, zweiten, dritten oder vierten Quadranten (sprich rechts oben, links oben, links unten oder rechts unten, vom Koordinatenursprung aus gesehen). Somit sollte die Bedeutung der Zeilen bis 1090 geklärt sein. Im folgenden muß ich kurz auf Ihr Wissen von trigonometrischen Funktionen zurückgreifen.

Im Programm (Zeilen 2130 bis 2150) werden solche Funktionen (wie SINus und COSinus) später verwendet. Prinzipiell gibt es zwei Gradeinheiten, mit denen man bei diesen Funktionen arbeiten kann, zum einen mit den gewohnten 360 Grad, zum anderen mit einer Einheit, die auf der berühmten Zahl Pi (π) aufbaut (genannt Bogenmaß; ich möchte hier nicht die mathematischen Grundlagen erläutern). Um die Gradeinheiten in das Bogenmaß umzurechnen, muß man die Grad mit Pi (π) multiplizieren und durch 180 dividieren. Und gerade dieser UmRechnungsfaktor (UR) steht in Zeile 1100.

Im weiteren Verlauf sollte die Bedeutung und Funktionsweise der Zeilen 2030 bis 2070 klar sein. Da dürfte es mittlerweile keine Probleme mehr geben.

In Zeile 2080 fragt der Computer nach der Höhe für Ihren Wurf (»WH« wie WurfHöhe). Zeile 2090 läßt nur Eingaben durch, die im erlaubten Bereich liegen, das heißt, sollte Ihre Eingabe unter eins oder über 90 liegen, werden Sie gleich noch einmal gefragt, so lange, bis Sie eine richtige Antwort eingeben.

Die nächsten drei Programmzeilen sind ganz analog aufgebaut, nur sind sie für den WurfWinkel (WW) zuständig. Der anschließende Teil ist für das Gesamtverständnis des

Programms nicht unbedingt erforderlich. Hier wird aus Ihren Eingaben nur errechnet, wo Ihr Schlammball hingeflogen ist. Um das Ganze nicht zu einfach werden zu lassen, wurden »komplizierte Verfahren« herangezogen. Hier werden Sie jetzt mit drei »neuen« Funktionen konfrontiert: ABS, SIN und COS.

ABS steht hier nicht für kürzeren Bremsweg, sondern für den »Absolutbetrag« einer Zahl. D.h., Sie bekommen immer nur Zahlen mit positivem Vorzeichen oder 0. Damit ersparen Sie sich eine Fallunterscheidung für den Fall, daß Sie ein negatives Ergebnis erhielten.

Beispiel:
PRINT ABS (-3)   ergibt 3.
PRINT ABS (5-10) ergibt 5.
PRINT ABS (534)   ergibt 534.
PRINT ABS (0)   ergibt 0.

Diese Funktion verändert also nichts am Wert der Zahl, sondern nur deren Vorzeichen. Die Funktionen SIN und COS brauche ich nicht zu erklären; ich baue da auf Vorkenntnisse aus dem (ehemaligen?) Mathematikunterricht. Neben diesen beiden Funktionen gibt es noch TAN für TANgens, ATN für ArcusTAngens etc. Ich möchte Sie hier auf den Anhang in diesem Buch verweisen; dort finden Sie eine Zusammenstellung aller Basic-Befehle sowie aller verfügbaren Funktionen). Falls Sie sich mit diesen beiden mathematischen Funktionen etwas auskennen, werden Sie auch sofort den Zusammenhang zwischen diesen Winkelfunktionen und den Koordinaten Ihres Wurfes verstehen, wenn Sie sich die kleine Tabelle in der Spielbeschreibung ansehen. Sie werden allerdings feststellen, daß die X-Koordinate eher dem Cosinus, und die Y-Koordinate dem Sinus folgt, also genau umgekehrt zu den beiden Zeilen 2140 und 2150; das beruht auf dem Faktor »AB«, mit dem die Werte jeweils multipliziert werden.

Wie der Abstand zwischen zwei Punkten berechnet wird, sehen Sie in Zeile 2160. Wenn Sie die mathematischen Grundlagen beherrschen (Satz des Pythagoras), dann bereitet Ihnen diese »Formel« keine allzugroßen Probleme. Wenn Sie diesen Satz nicht kennen, möchte ich Sie bitten, die Formel einfach so hinzunehmen. Es würde den Rahmen dieses Buches sprengen, wollte man die mathematischen Formeln einzeln erklären. Eine Funktion möchte ich aber noch kurz ansprechen, die SQR-Funktion. Sie steht für SQuare Root, zu deutsch Quadratwurzel. Das heißt, sie zieht aus dem Argument dahinter die Wurzel.

Beispiel:
PRINT SQR (4)   ergibt 2.

Dagegen dürften die Zeilen bis zum Ende eigentlich keine Schwierigkeiten mehr bereiten. Dennoch möchte ich auf das eine oder andere noch hinweisen: Zum Hauptprogramm (und von dort in den Endteil) wird nur verzweigt, wenn AS (AbStand Wurf - Drache) kleiner als 1000 ist (Zeile 2240). Wenn Sie also feststellen, daß Ihre Wurfkünste immer besser werden, und Sie mit diesen 999 Metern maximalem Abstand unterfordert sind, können Sie an dieser Stelle die Anforderungen höher schrauben. Sollten Sie die Anforderungen nicht erfüllen, müssen Sie noch einmal werfen, sprich, das Programm verzweigt zu Zeile 2030. Im Endteil ist nur eine Meldung vorgesehen, falls Sie mehr als neun Würfe gebraucht

haben. Sollten Sie sich hier und bereits an anderer Stelle gewundert haben, warum in diesem Fall neun und nicht zehn steht, so möchte ich das kurz erläutern: Die Abfrage in 3050 verzweigt, sobald die Bedingung `DG` (wie DurchGang) `<10` erfüllt ist. Das gilt für alle Zahlen von 1 bis 9. Das heißt also, sobald eine Zahl größer als neun vorkommt, ist diese Bedingung nicht mehr erfüllt.

Weiter oben war die Abfrage `IF AS<1000 THEN ...`: Hier gilt dasselbe. Etwas anderes wäre folgende Abfrage:

`IF AS<= 1000 THEN ...` (kleiner gleich)

Hier wäre die Bedingung auch noch für die Zahl 1000 erfüllt. Am Ende sehen Sie noch eine Neuerung (Zeile 3110). Sie starten die Programme doch immer mit RUN. Jetzt kommt dieser Befehl auch noch innerhalb eines Programmes vor. Und zwar wird gefragt, ob Sie noch einmal spielen möchten. Ist Ihre Antwort N(ein), kehrt das Programm mit RETURN zurück und trifft dann in Zeile 100 auf den Befehl END, der das Programm beENDet.

Was soll man aber tun, wenn das Programm fortgesetzt werden soll? Kehrt man mit RETURN zurück, wird es zwangsläufig abgebrochen, geht man mit einem GOTO-Befehl an den Beginn von dem Hauptteil SPIEL, gibt es gleich zwei Probleme: Zum einen hat der Drache wieder dieselben Koordinaten, und zum anderen kehrt er mit dem RETURN-Befehl am Ende doch zu Zeile 100 zurück; das heißt, das Programm wird dann bereits abgebrochen, ohne den Benutzer zu fragen, ob er nicht noch einmal spielen möchte.

Zu diesem zweiten Problem möchte ich noch etwas sagen. Und zwar gibt es in Ihrem C64 einen sogenannten Stapel (auf englisch auch »Stack«), in dem verschiedene Daten gespeichert werden. Sie als Basic-Programmierer haben keinen Zugang zu diesem Stapel. Dort werden unter anderem die Zeilennummern der GOSUB-Befehle gespeichert, an die der C64 nach einem RETURN wieder zurückspringen muß. Das heißt, in Zeile 70 trifft der Interpreter auf den ersten GOSUB-Befehl, im Stack steht jetzt 70 (etwas vereinfacht!). Danach springt der Computer, wie angegeben, zu Zeile 1000, um in Zeile 1110 auf RETURN zu treffen. Jetzt schaut er auf den Stapel, findet dort die Zeilennummer 70, springt zurück, findet dort keinen weiteren Befehl hinter dem GOSUB und springt daher in Zeile 80. Angenommen, wir befinden uns im Moment im Unterprogramm »ENDE«. Dann steht im Stack 90, da von dieser Zeile aus dieses Unterprogramm aufgerufen wurde. Wenn wir jetzt von hier aus zu Zeile 2030 verzweigen (mittels eines GOTO 2030), so bleibt der Stack davon unbeeinflußt. Sobald der C64 jetzt also auf ein RETURN trifft (das wäre in Zeile 2240), springt er zurück zu Zeile 90. In diesem Fall wäre das das kleinere Übel, da die unveränderten Koordinaten des Drachen Grisi sicher schwerer wiegen; aber dieses Hantieren mit dem Stack kann zu Problemen führen, während man es sich auf der anderen Seite auch wieder zu nutze machen kann.

4 Würfel

Einführung

Reizte es Sie nicht schon immer, einmal in einer Spielhölle Ihr Geld zu riskieren?

Ihr C64 kann Ihnen nur die Möglichkeit bieten, Ihr Geld beim Würfeln zu riskieren; die Atmosphäre und den wahren Nervenkitzel kann er natürlich nicht ins Haus bringen, zum einen fehlen da die Hintergrundgeräusche und der Zigarettenrauch, zum anderen spielen Sie nur mit fremdem Geld, nicht mit Ihrem eigenen.

Aber dennoch ist es ganz lustig, einmal sein Glück beim Würfeln zu versuchen. Allzuviel zu erklären gibt es bei diesem Programm allerdings nicht. Zunächst einmal muß ich die Regeln erklären. Sie würfeln (per Zufallsgenerator) mit zwei Würfeln. Zeigt die Gesamtaugenzahl sieben oder elf an, haben Sie Ihren Einsatz gewonnen.

Haben Sie zusammen zwei oder zwölf geworfen, werden Sie sofort »rausgeschmissen«, das heißt, Sie verlieren Ihren Einsatz und können wieder von vorne anfangen. Bei allen anderen Würfen wird Ihr Ergebnis festgehalten, und Sie müssen noch einmal würfeln. Haben Sie im zweiten Wurf zusammen sieben gewürfelt, verlieren Sie ebenfalls Ihren Einsatz. Nur wenn Sie dieselbe Augenzahl wie im ersten Wurf würfeln, gewinnen Sie Ihr gesetztes Geld.

In allen anderen Fällen, müssen Sie noch einmal würfeln. Dabei gelten dann wieder dieselben Regeln wie für diesen zweiten Wurf (rausgeschmissen bei sieben, Gewinn bei gleicher Augenzahl ...). Die Texte, die das Programm auf dem Bildschirm ausgibt, sind im übrigen selbsterklärend. Somit sollten Sie keine Probleme haben, sich »einzuspielen«.

Listing zu Würfel

```
10 REM ***************
20 REM ***         ***
30 REM *** WUERFEL ***
40 REM ***         ***
50 REM ***************
60 REM
70 GOSUB 1000
80 GOSUB 2000
90 GOSUB 3000
100 END
1000 REM
1010 REM *** VORBEREITUNGEN ***
1020 REM
1030 PRINT"{CLR,RVSON,40SPACE,RVOFF}";
1040 FOR I=1 TO 20
1050 :PRINT "{RVSON,SPACE,RVOFF,38SPACE,RVSON,SPACE,RVOFF}";
1060 NEXT
1070 PRINT"{RVSON,40SPACE,RVOFF}";
1080 GE=1500
1090 RETURN
1200 REM
1210 REM *** UNTERPROGRAMM 1 ***
1220 REM
1230 Z1=INT (RND(1)*6)+1
1240 Z2=INT (RND(1)*6)+1
1250 GOSUB 1300
1260 GET A$:IF A$="" THEN GOTO 1260
1270 RETURN
1300 REM
1310 REM *** UNTERPROGRAMM 2 ***
1320 REM
1330 POKE 211,11:POKE 214,10:SYS 58640
1340 ON Z1 GOSUB 1410,1450,1490,1530,1570,1610
1350 POKE 211,25:POKE 214,10:SYS 58640
1360 ON Z2 GOSUB 1410,1450,1490,1530,1570,1610
1370 RETURN
1410 PRINT "{3SPACE,DOWN,3LEFT}";
1420 PRINT " Q̲{SPACE,DOWN,3LEFT}";
1430 PRINT "{3SPACE}"
1440 RETURN
1450 PRINT "Q̲{2SPACE,DOWN,3LEFT}";
1460 PRINT "{3SPACE,DOWN,3LEFT}";
1470 PRINT "{2SPACE}Q̲"
1480 RETURN
1490 PRINT "Q̲{2SPACE,DOWN,3LEFT}";
1500 PRINT " Q̲{SPACE,DOWN,3LEFT}";
1510 PRINT "{2SPACE}Q̲"
1520 RETURN
1530 PRINT "Q̲ Q̲{DOWN,3LEFT}";
1540 PRINT "{3SPACE,DOWN,3LEFT}";
1550 PRINT "Q̲ Q̲"
1560 RETURN
1570 PRINT "Q̲ Q̲{DOWN,3LEFT}";
1580 PRINT " Q̲{SPACE,DOWN,3LEFT}";
1590 PRINT "Q̲ Q̲"
1600 RETURN
1610 PRINT "Q̲ Q̲{DOWN,3LEFT}";
1620 PRINT "Q̲ Q̲{DOWN,3LEFT}";
1630 PRINT "Q̲ Q̲"
1640 RETURN
1800 REM
1810 REM *** UNTERPROGRAMM 3 ***
1820 REM
1830 POKE 211,0:POKE 214,Y:SYS 58640
```

```
1840 PRINT "{40SPACE}";
1850 PRINT "{40SPACE}";
1860 RETURN
2000 REM
2010 REM *** EIGTL. SPIEL ***
2020 REM
2030 Y=22:GOSUB 1800
2040 Y=4:GOSUB 1800
2050 Y=6:GOSUB 1800
2060 PRINT "{HOME,3DOWN,3RIGHT}SIE HABEN{6SPACE,5LEFT}";GE;"{LEFT,SPACE}
     DM.{4SPACE}"
2070 POKE 214,23:SYS 58640
2080 INPUT "WIEVIEL MOECHTEN SIE RISKIEREN ";AN$
2090 RI=VAL(AN$)
2100 IF RI<0 OR RI>GE THEN GOTO 2030
2110 IF RI=0 THEN RETURN
2120 POKE 211,3:POKE 214,4:SYS 58640:PRINT "WETTE:";RI;"{LEFT,SPACE}"
2125 Y=22:GOSUB 1800
2130 POKE 211,9:POKE 214,22:SYS 58640:PRINT "(DIE WETTE LAEUFT...)"
2140 PRINT "{UP,SPACE}DRUECKEN SIE EINE TASTE UM ZU STOPPEN.";
2150 GOSUB 1200:SU=Z1+Z2
2160 Y=22:GOSUB 1800
2170 POKE 211,3:POKE 214,5:SYS 58640:PRINT "GEWUERFELT:";SU;"{LEFT,SPACE}"
2180 IF SU>3 AND SU<12 THEN GOTO 2230
2190 PRINT "{UP,3RIGHT}RAUSGESCHMISSEN ..."
2200 RI=-RI
2210 FOR PA=1 TO 1500:NEXT PA
2220 GOTO 2520
2230 IF SU<>7 AND SU<>11 THEN GOTO 2280
2240 POKE 211,3:POKE 214,7:SYS 58640:PRINT "SIE HABEN DIESEN WURF GEWONNEN !"
2250 FOR PA=1 TO 1500:NEXT PA
2260 GOTO 2520
2280 P=SU
2290 FOR PA=1 TO 1500:NEXT PA
2300 POKE 211,3:POKE 214,4:SYS 58640:PRINT "WETTE:";RI;" PUNKTE:";P
2310 Y=5:GOSUB 1800
2320 POKE 211,2:POKE 214,23:SYS 58640:PRINT "(WETTE LAEUFT ... TASTE DRU
     ECKEN)"
2330 GOSUB 1200
2340 SU=Z1+Z2
2350 POKE 211,3:POKE 214,6:SYS 58640:PRINT "{UP}GEWUERFELT ";SU
2360 IF SU<>7 THEN GOTO 2400
2370 PRINT "{UP,3RIGHT}RAUSGESCHMISSEN ..."
2380 FOR PA=1 TO 1500:NEXT PA
2390 GOTO 2490
2400 IF SU<>P THEN GOTO 2440
2410 PRINT "{UP,3RIGHT}SIE HABEN IHRE PUNKTE !!!"
2420 FOR PA=1 TO 1500:NEXT PA
2430 GOTO 2520
2440 PRINT "{UP,3RIGHT}SIE MUESSEN NOCH EINMAL WUERFELN ..."
2450 FOR PA=1 TO 1500:NEXT PA
2460 Y=22:GOSUB 1800
2470 Y=5:GOSUB 1800
2480 GOTO 2320
2490 GE=GE-RI
2500 IF GE<=0 THEN RETURN
2510 GOTO 2030
2520 GE=GE+RI
2530 GOTO 2030
3000 REM
3010 REM *** ENDE ***
3020 REM
3030 PRINT "{CLR}"
```

```
3040 POKE 214,10:SYS 58640
3050 PRINT "SIE HOERTEN MIT";GE;"DM AUF."
3060 IF GE<100 THEN PRINT "UEBUNG, UEBUNG !":RETURN
3070 IF GE<500 THEN PRINT "IMMERHIN ...":RETURN
3080 IF GE<1000 THEN PRINT "NICHT SCHLECHT !":RETURN
3090 PRINT "JUNGE, JUNGE !":RETURN
```

■ Programmbeschreibung für Würfel

Obwohl die »Vorbereitungen« diesmal nicht so schwer sind, möchte ich doch ein paar Worte darüber verlieren. Im großen und ganzen wird lediglich eine besondere Ausgabe vorbereitet. Dafür wird um den Bildschirm ein Rahmen gezogen, und das macht das Unterprogramm ab Zeile 1030.

In der ersten Zeile wird eine inverse Zeile (sprich in der Zeichenfarbe) gezogen; Sie erinnern sich doch: [CTRL]+[9] ergibt ein großes, inverses R, das für »revers on« steht. Mit der FOR-NEXT-Schleife wird in jeder der 20 folgenden Zeilen jeweils am linken und rechten Rand eine Spalte invers »gezeichnet«. Und abschließend wird in Zeile 1070 noch einmal eine komplette Zeile invers ausgedruckt. Am Ende jeder Ausgabezeile (1030, 1050 und 1070) muß jeweils ein Strichpunkt stehen. Wenn Sie mir das nicht glauben, entfernen Sie doch diese Strichpunkte und lassen das Programm noch einmal laufen; spätestens dann werden Sie mir glauben, daß die Strichpunkte notwendig sind.

Zuletzt wird Ihr Geldvorrat mit 1500 GeldEinheiten (GE) festgelegt. Wenn Sie das Listing weiterverfolgen, sehen Sie jetzt drei weitere Unterprogramme. Sie werden aus dem Programmteil »EIGTL. SPIEL« aufgerufen. Dennoch möchte ich sie gleich hier behandeln.

In Unterprogramm 1 werden zwei Zufallszahlen bestimmt. Diese beiden Zahlen stellen Ihren Wurf dar. Außerdem wird noch dafür gesorgt, daß beide Zahlen ausgegeben werden, indem ein Aufruf des Unterprogramms 2 erfolgt. Danach wartet die Routine 1 noch auf einen Tastendruck, bevor sie mit RETURN wieder zurückkehrt.

Wie gesagt, gibt Unterprogramm 2 beide Zahlen (Z1 und Z2) aus. In den ersten zwei Zeilen kommen gleich einige neue Sachen auf Sie zu: Da wäre zum Beispiel der POKE-Befehl. Um ihn zu erklären muß ich weit ausholen (machen Sie es sich schon einmal bequem!).

Wie Sie wissen, bietet Ihnen der C64 64 Kilobyte (abgekürzt Kbyte) Speicher (genaugenommen sind es 65535 Byte). Davon stehen Ihnen 38 Kbyte für Basic zur Verfügung. Großteile des restlichen Speichers sind belegt, zum einen vom Betriebssystem, und zum anderen vom Interpreter. Man kann nun den Speicher Ihres C64 in 65535 einzelne Speicherstellen aufteilen. Und jede dieser Speicherstellen kann Werte zwischen 0 und 255 annehmen, das ergibt also 256 Möglichkeiten. Mit dem POKE-Befehl können Sie einer solchen Speicherstelle einen bestimmten Wert zuschreiben.

Beispiel:

```
POKE 1024,3
```

In diesem Fall hat die Speicherstelle 1024 den Wert drei. Man muß aber genau wissen, welcher Speicherstelle man welchen Wert zuordnet. Falsche Werte an falschen Adressen können verheerende Auswirkungen haben. Wie gesagt, sind bestimmte Speicherbereiche in Ihrem C64 bereits belegt. So steht beispielsweise an den Adressen 1024 bis 2023 der Bildschirmspeicher. Jede einzelne Speicheradresse steht für eine Spalte in einer Zeile. Wenn Sie das Beispiel von vorhin einmal ausprobieren, erscheint in der linken oberen Ecke ein »C«. Nun zu unserem Fall in Zeile 1330. In Speicherstelle 211 steht der X-Wert, in 214 der Y-Wert des Cursors. Wenn Sie diese Speicherstellen direkt mittels des POKE-Befehls manipulieren, und direkt anschließend einen Text mit PRINT ausgeben lassen, wird dieser Text an der angegebenen Cursor-Position erscheinen. Doch vor der Ausgabe des Textes muß dem C64 noch mitgeteilt werden, daß sich die Werte des Cursors verändert haben. Dazu müssen Sie eine Routine (also eine Art Unterprogramm) im Betriebssystem Ihres C64 aufrufen. Dieses Betriebssystem ist in Maschinensprache geschrieben, und Maschinenspracheprogramme werden von Basic aus mit einem SYS-Befehl aufgerufen. Jetzt müssen Sie dem C64 nur noch mitteilen, ab welcher Adresse das Unterprogramm steht, in unserem Fall ab Adresse 58640. Wenn diese Routine beendet ist, kehrt der C64 selbständig wieder zu der Zeile zurück, von der aus das Unterprogramm aufgerufen wurde (funktioniert ähnlich wie der GOSUB-Befehl).

Das heißt, Zeile 1330 ist für die Steuerung des Cursors zuständig, genauso wie Zeile 1350. Sie werden in diesem und den folgenden Programmen noch häufiger über diese Cursor-Steuerung stolpern. Sie ist ein ziemlich bequemes Mittel, um einen etwas interessanteren Bildschirmaufbau zu erhalten.

Aber der Streß geht gleich weiter, denn in Zeile 1340 (und 1360) kommt schon wieder ein neuer Befehl auf Sie zu: ON ... GOSUB ... oder ON ... GOTO ...

Zwischen ON und GOSUB/GOTO wird in den meisten Fällen eine Variable stehen; nach dem GOSUB/GOTO folgt eine ganze Liste von Zeilennummern, an die der Computer springen soll, kann, darf, muß, will ...

Doch bleibt es nicht der Willkür Ihres C64 überlassen, an welche Zeile er springen soll. Vielmehr entsprechen die einzelnen Zeilennummern den möglichen Inhalten der Variable. In unserem Beispiel kann die Variable Z1 die Werte 1 bis 6 annehmen. Da die Ausgabe so programmiert wurde, daß für jede Zahl ein Würfel nachgebildet wird, muß also für jede Ziffer eine andere Ausgabe vorhanden sein. Und diese Unterscheidung läßt sich ganz einfach mit dem ON-Befehl schaffen. Ist Z1=1, verzweigt der Computer zu Zeile 1410, Z1=2 Zeile 1450, Z=3 Zeile 1490 etc.

Ein weiteres Beispiel wäre ein Menü bei einer Adreßverwaltung: Angenommen, Sie können zwischen »Daten eingeben«, »Daten speichern«, »Daten laden« und »Daten drucken« wählen. Somit haben Sie vier Punkte, zwischen denen Sie wählen können. Wenn Sie jetzt jedem Menüpunkt eine der Ziffern von 1 bis 4 zuordnen, und der Anwender dann

diese Zahl eingibt, können Sie mit dem ON-Befehl ganz einfach zu den einzelnen Routinen springen, die den Menüpunkten entsprechen.

Eines möchte ich noch anmerken. Angenommen, Sie benutzen den ON-Befehl in Verbindung mit GOSUB, dann springt der C64 beim nächsten RETURN nicht zur folgenden Zeilennummer in der Liste, sondern zum nächsten Befehl.

Beispiel: Angenommen, Z1 ist 1. Das bedeutet, der C64 springt zur Routine ab Zeile 1410, stößt in 1440 auf ein »RETURN« und kommt zurück. Er findet in der gemerkten Zeile 1340 keinen weiteren Befehl und geht zu Zeile 1350. Er nimmt also nicht die nächste Nummer in der Liste (in dem Fall 1450), springt dorthin, danach die nächste Zeilennummer usw.

Alles in allem steuern die Zeilen 1330 und 1340 die Ausgabe für »Z1«, während 1350 und 1360 die für »Z2« steuern. Beide Teile verwenden dieselben Unterroutinen (ab Zeile 1410); der einzige Unterschied ist, daß die zweite Ausgabe um 14 Zeichen weiter rechts auf dem Bildschirm erfolgt (beachten Sie den Wert hinter POKE 211...).

Abschließend möchte ich Sie noch darauf aufmerksam machen, daß man mehrere GOSUB-Befehle ineinanderschachteln kann, zum Beispiel:

```
 10 GOSUB 50
 20 END
 50 GOSUB 100
 60 RETURN
100 GOSUB 150
110 RETURN
150 RETURN
```

So ähnlich funktioniert das Programm Würfel:

Zeile 80 ruft Zeile 2000 auf.
: Aus dieser Unterroutine wird Unterprogramm 1 aufgerufen.
:   Unterprogramm 1 ruft Unterprogramm 2 auf.
:   Innerhalb von Unterprogramm 2 wird die Ausgaberoutine aufgerufen.

Sie sehen also, wie verschachtelt das Programm hier arbeitet. Nun noch ein Wort zu den kleinen Ausgaberoutinen, die jeweils die Würfel darstellen sollen. Die Cursor-Position liegt bereits fest. Diese wurde vor dem Aufruf manipuliert. Wenn die oberste Zeile des Würfels dargestellt ist, muß das Programm den Cursor um eine Zeile nach unten und um drei Zeichen nach links lenken, damit die zweite Zeile schön unter die erste kommt. Diese Steuerung können Sie an den inversen Zeichen immer am Ende der ersten und zweiten Ausgabezeile erkennen (zum Beispiel Zeilen 1410, 1420). Vielleicht fragen Sie sich, warum die oberste Zeile immer mitausgegeben wird, obwohl sie beispielsweise bei der Ziffer 1 leer ist. Wenn Sie das Programm einige Male gespielt haben, werden Sie wissen, warum. Da das Programm immer wieder denselben Bildschirmaufbau verwendet, kann es sein, das eine solche Ausgabe »über« eine alte, vorher erfolgte Ausgabe gelegt wird. Würde man daher die freien Bereiche nicht löschen, würden die alten Ziffern »durchschimmern«, sprich, man könnte bald nichts mehr erkennen. Wenn Sie in diesen Ausgaberoutinen einmal jedes [SPACE] (Leertaste) durch ein [CRSR-RIGHT] ersetzen,

werden Sie die Auswirkungen verstehen (vergessen Sie dabei aber nicht, vor den [CRSR-RIGHT] noch einmal das Anführungszeichen einzugeben, da Sie sonst nicht die inversen Zeichen dafür sehen!).

Nach diesem etwas theoretischen Ausflug geht es weiter zu Unterprogramm 3. Wie gesagt, benutzt das Programm denselben Bildschirmaufbau immer wieder. Um kein Chaos entstehen zu lassen, müssen daher bestimmte Bereiche des Bildschirms hin und wieder gelöscht (mit Leerzeichen überschrieben) werden. Diese Routine löscht zwei Zeilen. Wenn man der Variablen »Y« einen Wert zuweist, werden die Bildschirmzeile »Y« und die darunterliegende gelöscht.

In den Zeilen 2030 bis 2050 sehen Sie gleich die Anwendung dieser Routine. Da werden die Zeilen 22, 23, 4, 5, 6 und 7 gelöscht. Anschließend erfolgt die Ausgabe, wieviel Geld Sie (noch übrig) haben.

Manchmal (eigentlich meistens) erübrigt es sich, die X-Koordinate noch einmal auf 0 zu setzen. Man sollte es vielleicht der Ordnung halber tun. An dem einen Befehl wird man sich nicht zu Tode tippen.

In Zeile 2080 werden Sie nach einem Geldbetrag gefragt, nämlich nach dem, den Sie gerne riskieren möchten. Beachten Sie bitte, daß Ihre Eingabe in einem String gespeichert wird. Dazu möchte ich noch ein paar Anmerkungen machen: Schreiben Sie ein ganz einfaches Programm, das lediglich aus einer Zeile besteht und von Ihnen eine Eingabe mittels des INPUT-Befehls erwartet. Setzen Sie als Variable eine ganz normale Zahlenvariable (zum Beispiel A) ein. Wenn Sie das Programm laufen lassen und bei der Eingabe einen (oder mehrere) Buchstaben eingeben, werden Sie eine Fehlermeldung erhalten und aufgefordert werden, Ihre Eingabe noch einmal zu tätigen. Wenn Sie aber ein Programm schreiben möchten, das solche Fehleingaben abfängt, empfiehlt es sich, die Eingabe in einen String zu übernehmen und dann die Eingabe auszuwerten. Man kann dann gegebenenfalls eine Fehlermeldung auf dem Bildschirm ausgeben lassen und den Benutzer höflich darauf aufmerksam machen, daß er nur zugelassene Zeichen und Zahlen benutzen solle.

Wenn Sie die Eingabe (sagen wir einen Geldbetrag) in einem String gespeichert haben und mit dieser Eingabe nun weiterrechnen wollen, gibt es zwei mögliche Wege: Entweder wandeln Sie die Zahl in dem String in eine reine Zahl um, oder Sie arbeiten mit dem String weiter. Dazu müssen die IF-Abfragen etwas anders aussehen:

Beispiel:

```
... IF A$="1" THEN ...
```

Sie sehen also, daß die 1 nicht als Zahl, sondern als Zeichen aufgefaßt wird. Im anderen Fall – also wenn Sie den String umwandeln wollen – können Sie Funktionen des C64 in Anspruch nehmen. Er bietet Ihnen die VAL-Funktion, die aus einem String eine Zahl macht; das funktioniert allerdings nur, wenn in dem String nichts als Ziffern gespeichert sind.

Beispiel

```
A=VAL (A$)
```

Angenommen, »A$« enthielt die Zeichenkette »1234«, dann enthält »A« nach dieser Zeile den Wert 1234. Sollte »A$« aber auch Buchstaben enthalten haben (A$="12df4"), dann funktioniert diese Funktion nicht mehr. Ein letztes Beispiel möge diesen Befehl abschließen:

```
ZU=VAL (WE$)
```

Zurück zu unserem Programm: Da wurde Ihr Geldbetrag in dem String »AN$« (wie ANtwort) gespeichert. In die Variable »RI« (RIsiko) wird der Wert übertragen (Zeile 2090). Mit geringem Aufwand können Sie Ihre Programme ein bißchen sicherer gestalten, so daß nicht jeder noch so kleine Fehler das Programm unbenutzbar macht. Sollte Ihre Eingabe unsinnig gewesen sein, so kehrt das Programm in Zeile 2100 zur Eingabe zurück. Haben Sie 0 als Risiko eingegeben, wird das Programm abgebrochen (Zeile 2110).

Sehen Sie sich jetzt das Ende von Zeile 2120 an. Ich nehme an, Sie verstehen nicht sofort, was der Rest hinter der Variablen »RI« für einen Sinn hat; Sie werden den Sinn aber sofort begreifen, wenn Sie diesen Teil einmal entfernen. Da auch dieses Programm immer wieder denselben Bildschirmaufbau verwendet, werden bestimmte Teile jedesmal überschrieben. Nehmen Sie an, Sie hätten im ersten Spiel 1000 Mark riskiert, im zweiten aber nur 300. Da das Programm die 1000 mit 300 überschreibt, ohne den Rest zu löschen, sähen Sie 3000 Mark als gesetzt. Um das zu verhindern, wird der Teil hinter der aktuellen Ausgabe gelöscht. Das ist auch der Grund für das komische Aussehen von Zeile 2060. Eins muß ich aber noch dazu sagen: Wenn Sie in einer Zeile mit einem PRINT-Befehl direkte Bildschirmausgabe und Ausgabe von Variablen mischen, fügt der C64 dazwischen automatisch ein Leerzeichen ein.

Beispiel:

```
A=5: PRINT "ICH HABE";A;"GESCHWISTER."
```

Die Ausgabe auf dem Bildschirm sieht folgendermaßen aus:

```
ICH HABE 5 GESCHWISTER.
```

Darum wird hinter das »RI« in Zeile 2120 (und in Zeile 2170 hinter »SU«) noch ein [CRSR-LEFT] gesetzt, damit wirklich von der letzten Ziffer ab gelöscht wird. Sollten diese Ausführungen etwas theoretisch für Sie sein, so kann ich Ihnen nur empfehlen, testen Sie, probieren Sie herum; sollte ein Programm anschließend gar nicht mehr laufen, können Sie es immer wieder von Diskette laden, solange Sie nicht die veränderte Version gespeichert haben.

Zeile 2125 wurde nachträglich eingefügt, wie Sie an der Zeilennummer sehen; dadurch sollte die Bildschirmdarstellung verbessert werden; und zwar werden die Bildschirmzeilen 22 und 23 gelöscht, um gleich danach (Zeilen 2130, 2140) wieder mit Text beschrieben zu werden; aber auf diese Art und Weise verhindert man wieder Überschneidungen von Text.

In Zeile 2150 wird Unterprogramm 1 aufgerufen; das heißt, der Computer setzt zwei Zufallszahlen fest, gibt sie aus und wartet auf einen Tastendruck (siehe oben). Sobald der

Interpreter wieder in Zeile 2150 angekommen ist, werden die beiden Einzelzahlen addiert und in »SU« (wie SUmme) gespeichert.

Wieder werden die Bildschirmzeilen 22 und 23 gelöscht, und die Gesamtaugenzahl wird in der Bildschirmzeile 5 ausgegeben. Hier beginnt jetzt die Auswertung Ihres »Wurfes«. Dazu wird zuerst überprüft, ob Ihre Summe zwischen drei und zwölf liegt. Ist das der Fall verzweigt der Computer zu Zeile 2230 für weitere Überprüfungen; andernfalls (das heißt SU=2, 3, oder 12) werden Sie »rausgeschmissen« (Zeilen 2190-2220). Dafür bekommt Ihr Einsatz »RI« ein negatives Vorzeichen, und - nachdem eine kleine Pause verstrichen ist - springt der C64 zu Zeile 2520, um dort Ihren Einsatz zu Ihrem Guthaben zu addieren; und da Ihr Einsatz jetzt negativ ist, wird praktisch subtrahiert.

Bei der weiteren Überprüfung wird getestet, ob Sie eine Sieben oder eine Elf gewürfelt haben. Falls dem so ist, haben Sie den Wurf gewonnen (siehe Zeilen 2240-2260). In allen anderen Fällen müssen Sie noch einmal würfeln (ab Zeile 2280). Dazu wird die Summe, die Sie im ersten Wurf erreicht haben in der Variablen »P« zwischengespeichert, damit dieser Wert später zum Vergleich zur Verfügung steht. Bis Zeile 2350 verläuft das Programm jetzt ganz analog zu den Erklärungen oben.

Sollten Sie im zweiten Wurf eine Sieben gewürfelt haben, werden Sie auf alle Fälle »rausgeschmissen« (Zeilen 2360-2390). Andernfalls wird geprüft, ob Sie dieselbe Augenzahl erreicht haben wie im ersten Wurf. Ist das der Fall, erhalten Sie Ihren Einsatz zu Ihrem Guthaben dazu (Zeilen 2400-2430). Und in allen anderen Fällen, müssen Sie noch einmal würfeln, sprich so lange, bis Sie eine Sieben oder Ihre vorherige Augenzahl gewürfelt haben.

Zu Zeile 2490 wird von 2390 verzweigt, wenn Sie also verloren haben, und Ihnen Ihr Einsatz abgezogen wird. Dagegen wird in Zeile 2520 der Einsatz jeweils addiert. Natürlich hätte man sich die Zeile 2200 sparen können und anstatt in 2220 nach 2520 zu springen, hätte man auch nach 2490 verzweigen können, um Ihren Einsatz abzuziehen. Aber ich wollte Ihnen einmal zeigen, wie vielfältig selbst solche Kleinigkeiten gelöst werden können.

Ich glaube, die restlichen Programmzeilen kann ich getrost übergehen. Sie sollten sich diesen Teil aber noch zu Gemüte führen. Und beachten Sie bitte, daß jedes einzelne RETURN in den Zeilen 3060 bis 3090 notwendig ist. Warum? Wenn Sie das nicht mehr wissen, probieren Sie es doch einmal ohne diese RETURNs aus!

5 Snake

Einführung

Dieses Programm ist mehr oder weniger als Erholung gedacht. Stellen Sie sich vor, Sie sitzen in Ihrer zukünftigen Programmiererkarriere stundenlang vor dem Bildschirm und sehen nichts als Programmzeilen und Befehle ...

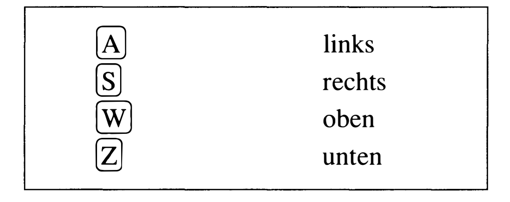
Da ist jedes noch so einfache, aber kurze Spiel willkommen. Und diesen Bedarf soll Snake decken. Jedoch sollten Sie einen wohlgesinnten Mitspieler bei sich haben. Mitspieler deshalb, weil das Spiel für zwei Spieler ist; wohlgesonnen, da immer ein Spieler – je nach Tastenverteilung – bevorteilt ist.

In diesem Spiel gibt es zwei Schlangen, die versuchen müssen, einander ihr Territorium (hier den Bildschirm) streitig zu machen. Die beiden Spieler steuern ihre Schlange jeweils mit vier Buchstaben (für jede Richtung eine Taste). Dabei wird die Schlange bei jedem Schritt um ein Glied länger. Jetzt müssen Sie versuchen, ein möglichst großes Gebiet mit Ihrer Schlange zu belegen.

Eine andere mögliche Taktik ist es, den Gegner einzukreisen. Dazu muß man wissen, daß das Spiel beendet ist, sobald ein Spieler den anderen oder den Bildschirmrand berührt. Dabei verliert derjenige, der die Berührung ausgelöst hat.

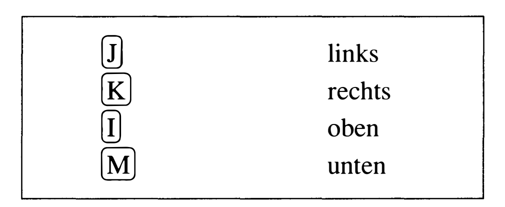
Daher kann es gut sein, daß der andere Spieler nichts tut und auf einen Fehler Ihrerseits wartet – und im Endeffekt vielleicht sogar gewinnt! Wie Sie sich also taktisch verhalten, bleibt Ihnen überlassen. Nun aber noch ein Wort zur Tastatursteuerung:

Spieler 1 bewegt seine Schlange mit:



Taste	Richtung
A	links
S	rechts
W	oben
Z	unten

Für Spieler 2 gilt:



Taste	Richtung
J	links
K	rechts
I	oben
M	unten

Bei dieser Anordnung ist Spieler 2 immer etwas im Vorteil, da seine Tasten vom Computer bevorzugt erkannt werden. Auf alle Fälle wünsche ich viel Spaß beim Herumexperimentieren – und ich glaube, gerade dieses Programm lädt dazu ein; zum einen, weil es sehr kurz und übersichtlich ist, und zum anderen, weil es stark »verbesserungsbedürftig« ist; aber das ist Ihre Sache!

Listing zu Snake

```
10 REM *************
20 REM ***       ***
30 REM *** SNAKE ***
40 REM ***       ***
50 REM *************
60 REM
70 GOSUB 1000
80 GOSUB 2000
90 GOSUB 3000
100 END
1000 REM
1010 REM *** VORBEREITUNGEN ***
1020 REM
1030 X1=INT(RND(1)*5)+1
1040 Y1=INT(RND(1)*6)+6
1050 X2=INT(RND(1)*5)+34
1060 Y2=INT(RND(1)*6)+6
1070 PRINT "{CLR}":POKE 650,128
1080 RETURN
2000 REM
2010 REM *** SPIEL ***
2020 REM
2030 POKE 211,X1:POKE 214,Y1:SYS 58640
2040 PRINT "{BLACK}*";
2050 POKE 211,X2:POKE 214,Y2:SYS 58640
2060 PRINT "{WHITE}O";
2090 GET A$
2100 IF A$="W" THEN Y1=Y1-1:GOTO 2190
2110 IF A$="Z" THEN Y1=Y1+1:GOTO 2190
2120 IF A$="A" THEN X1=X1-1:GOTO 2190
2130 IF A$="S" THEN X1=X1+1:GOTO 2190
2135 :
2140 IF A$="I" THEN Y2=Y2-1:GOTO 2210
2150 IF A$="M" THEN Y2=Y2+1:GOTO 2210
2160 IF A$="J" THEN X2=X2-1:GOTO 2210
2170 IF A$="K" THEN X2=X2+1:GOTO 2210
2180 GOTO 2090
2190 IF X1<0 OR X1>39 OR Y1<1 OR Y1>24 THEN SG=2:RETURN
2200 IF PEEK(1024+X1+Y1*40)<>32 THEN SG=2:RETURN
2205 GOTO 2030
2210 IF X2<0 OR X2>39 OR Y2<1 OR Y2>24 THEN SG=1:RETURN
2220 IF PEEK(1024+X2+Y2*40)<>32 THEN SG=1:RETURN
2230 GOTO 2030
3000 REM
3010 REM *** ENDE ***
3020 REM
3030 PRINT "{CLR}":POKE 211,7:POKE 214,10:SYS 58640
3040 PRINT "DAS SPIEL IST BEENDET ..."
3050 PRINT "{3DOWN,7SPACE}SPIELER";SG;"HAT GEWONNEN !"
3060 RETURN
```

■ Programmbeschreibung für Snake

Seien Sie beruhigt: Die Erläuterungen zu diesem Spiel können gar nicht so lang werden, wie die vorhergehenden; dazu reicht schon die Programmlänge bei weitem nicht aus. In den ersten vier Zeilen ab 1030 werden jeweils zwei X- und Y-Werte zufällig ausgewählt. Die X1/Y1-Koordinaten geben an, von wo aus Spieler 1 startet; das Gleiche gilt analog für Spieler 2.

Wundern Sie sich bitte nicht über die scheinbar so »krummen« Werte bei den Zufallsfunktionen. Versuchen Sie vielmehr, selbst auf das »Geheimnis« dieser Zeilen zu kommen!

Vielleicht hilft es, wenn ich verrate, daß keine beliebigen Werte herauskommen sollen, sondern nur solche, die in einem ganz engen Intervall liegen.

Wahrscheinlich verrate ich jetzt nichts Neues, wenn ich preisgebe, daß »X1« Werte zwischen eins und fünf, »Y1« solche zwischen sechs und elf, »X2« zwischen 34 und 38 und »Y2« wieder zwischen sechs und elf annehmen kann. Damit soll gewährleistet werden, daß die beiden Schlangen nicht schon zu Beginn des Spiels zu sehr aufeinander hängen.

In Zeile 1070 wird der Bildschirm gelöscht. Doch der Befehl danach ist neu. Dem POKE-Befehl sind Sie zwar schon begegnet, aber nicht mit der Adresse 650. Diese Adresse beeinflußt die Tastatur. Mit dem angegebenen POKE-Befehl werden alle Tasten auf Dauerbetrieb umgeschaltet. Sie wissen doch, daß normalerweise nur solche Sondertasten wie [SPACE], [INST/DEL] und andere desto länger funktionieren, je länger man sie betätigt. Buchstabentasten dagegen geben einen Buchstaben aus und sprechen dann nicht weiter an. Das kann man mit diesem Befehl umgehen. Wenn Sie ihn eingeben und anschließend zum Beispiel [A] gedrückt halten, können Sie mit einmaligem (dafür langen) Drücken den gesamten Bildschirm mit »A«s füllen.

Sobald das Programm aus den Vorbereitungen zurückgekehrt und wieder in das Hauptprogramm verzweigt ist, werden dort die Koordinaten für Spieler 1 und 2 gesetzt. Dazu werden noch die Erkennungszeichen für beide Spieler ausgegeben. In Zeile 2090 erwartet das Programm eine Eingabe. Allerdings wird hier keine Leereingabe abgefangen.

Daraufhin wird die Eingabe ausgewertet. Je nach Tasten werden die Koordinaten von Spieler 1 verändert; beispielsweise wird bei [W] die Y-Koordinate um eins vermindert. Dazu muß man wissen, daß ein Koordinatensystem beim C64 prinzipiell seinen Ursprung in der linken oberen Bildschirmecke hat. Daher muß in dem Fall »Y« um eins verkleinert werden. Entsprechendes gilt für die X-Koordinate.

Jedesmal, wenn eine Koordinate verändert wurde, springt der Computer anschließend zu Zeile 2190, um zu überprüfen, ob eine Kollision oder ähnliches stattgefunden hat. Doch dazu später mehr. Sollte Spieler eins keine Taste gedrückt haben, das heißt also, daß das Programm die Zeilen ab 2135 überhaupt abarbeitet, werden die Koordinaten des zweiten Spielers entsprechend geändert.

Die Zeile 2135 hat keine spezielle Programmfunktion. Sie soll lediglich die Übersichtlichkeit des Programms erhöhen. Sie können ja einmal ausprobieren, wie gut (bzw. schlecht) sich die Abfragen in den Zeilen 2100 bis 2170 ohne 2135 auseinanderhalten lassen. Eine ähnliche Funktion übernimmt übrigens in jedem Programm die Zeile 60. Sie soll den »Programmkopf« vom »Rumpf« trennen. Nötig wäre sie nicht, aber ich möchte sie auch nicht unbedingt als überflüssig bezeichnen.

Sollte keine der Tasten gedrückt worden sein, die für die Steuerung relevant sind, kommt das Programm zu Zeile 2180 und springt in jedem Fall zu Zeile 2090. Das gilt auch für den Fall, daß gar keine Eingabe vorlag. Daher konnte also die Abfrage auf eine Leereingabe in 2090 weggelassen werden.

In Zeile 2190 wird zunächst überprüft, ob die Koordinaten von Spieler 1 den Rand berühren. Sollten sie außerhalb des erlaubten Bereichs sein (eine der vier Bedingungen in der IF-Abfrage ist erfüllt), wird die Variable »SG« (Spieler Gewinnt) auf 2 gesetzt.

Sollte dagegen das Feld, das Spieler 1 im Moment betreten hat, nicht frei sein, wird diese Gewinnervariable auch auf zwei gesetzt; in beiden Fällen kehrt der Interpreter mit »RETURN« zurück, um den Programmteil ab Zeile 3000 abzuarbeiten. Zu der Überprüfung, ob die Stelle frei ist, möchte ich noch etwas sagen: Sie haben schon früher erfahren, daß der Bildschirmspeicher Ihres C64 in den Speicherstellen 1024 bis 2023 liegt (25 Zeilen zu je 40 Zeichen).

Damals haben Sie gelernt, eine Speicherstelle mittels »POKE« zu verändern (siehe auch Zeilen 2030 und 2050). Jetzt lernen Sie eine neue Funktion kennen, mit dem man den Inhalt einer Speicherstelle erfahren kann: die PEEK-Funktion.

Wenn Sie beispielsweise den Bildschirm löschen und danach im Direktmodus den Befehl:

```
PRINT PEEK (1024)
```

eingeben, werden Sie als Ergebnis 32 erhalten. Geben Sie jetzt POKE 1024,1 ein; in der oberen linken Ecke sollte ein »A« stehen. Wenn Sie jetzt noch POKE 1024,32 eingeben, müßte das »A« wieder verschwinden. Dasselbe können Sie mit jeder anderen Speicherstelle zwischen 1024 und 2023 ausprobieren. Sie können hinter den POKE-Befehl auch beliebige andere Zahlen außer eins und 32 setzen; die einzige Bedingung, die gestellt wird, ist, daß die Zahl zwischen 0 und 255 liegen muß.

Wie gesagt, können Sie mit »PEEK« den Inhalt einer Speicherstelle erhalten. In unserem Fall ist der Inhalt der Adresse (Speicherstelle) gefragt, die durch »X1« und »Y1« bzw. »X2« und »Y2« repräsentiert wird. Man kann nicht einfach 1024 + X1 + Y1 schreiben, das ergäbe für X1=2 und Y1=2 die Speicherstelle 1028; die steht aber für die Koordinaten X1=4, Y1=0.

Man muß also die Y-Koordinate noch mit der Zahl der Spalten pro Zeile multiplizieren, um wirklich in die richtige Zeile zu kommen (beim C64 sind das 40 Zeichen pro Zeile). Jetzt verstehen Sie auch den Aufbau dieser PEEK-Funktion. Wie Sie oben gesehen haben,

ergibt »PEEK« für eine freie Bildschirmspalte den Wert 32. Sollte der ermittelte Wert davon abweichen, bedeutet das, daß diese Stelle nicht frei ist, der Spieler also verloren hat.

Für den Spieler zwei gilt dasselbe ganz analog; nur wird bei ihm die Variable »SG« nicht auf zwei, sondern auf eins gesetzt.

Die restlichen vier Zeilen kann ich mir doch wohl sparen, oder?

6 Zahlendreher

Einführung

In diesem Spiel sollen Sie eine Liste von Zahlen, die der Computer verdreht hat, wieder in die richtige Reihenfolge bringen. Ganz so einfach, wie es klingt, geht es natürlich nicht! Sie können die Liste immer nur ab einer bestimmten Position drehen. Es dürfte am einfachsten sein, wenn ich Ihnen dieses System anhand eines Beispiels erkläre. Andernfalls wird es zu theoretisch:

Nehmen wir als Beispiel folgende Liste:

```
Position:   0 1 2 3 4 5 6 7 8 9
Liste:      0 1 9 4 5 6 3 2 7 8
```

Die Position gibt an, ab welcher Stelle die Liste gedreht werden soll. Gleichzeitig sieht so die Liste aus, wenn Sie sie entwirrt haben. Nehmen wir an, Sie drehen die Liste ab Position sieben:

```
Position:   0 1 2 3 4 5 6 7 8 9
Liste:      0 1 9 4 5 6 3 8 7 2
```

Das Prinzip funktioniert also folgendermaßen: Die erste Zahl und die letzte werden miteinander vertauscht, dann sind die zweite und die vorletzte Zahl an der Reihe ...

Sollte die Anzahl dieser Zahlen ungerade sein, bleibt die mittlere Ziffer an ihrer Stelle. Abschließend drehen wir die obige Liste noch einmal, diesmal ab Position 2:

```
Position:   0 1 2 3 4 5 6 7 8 9
Liste:      0 1 2 7 8 3 6 5 4 9
```

Verstehen Sie jetzt das Prinzip? Sie können ja mit dem Programm ein wenig herumexperimentieren, um mit diesem System vertraut zu werden. Ich habe bei diesen Beispielen nicht versucht, die Liste zu lösen; das überlasse ich Ihnen.

Listing zu Zahlendreher

```
10 REM *******************
20 REM ***             ***
30 REM *** ZAHLENDREHER ***
40 REM ***             ***
50 REM *******************
60 REM
70 GOSUB 1000
80 GOSUB 2000
90 GOSUB 3000
100 END
1000 REM
1010 REM *** VORBEREITUNGEN ***
1020 REM
1030 DIM PO (9)
1040 FOR I=0 TO 9:PO (I)=I:NEXT I
1050 FOR I=0 TO 9
1060 :X=INT (RND (1)*10)
1070 :H=PO(I)
1080 :PO(I)=PO(X)
1090 :PO(X)=H
1100 NEXT I
1110 RETURN
2000 REM
2010 REM *** SPIEL ***
2020 REM
2030 PRINT "{CLR,2DOWN}";TAB(13);"ZAHLENDREHER"
2040 PRINT "{3DOWN}POSITION: 0 1 2 3 4 5 6 7 8 9"
2050 PRINT TAB(10);"-------------------"
2060 PRINT "LISTE:{3SPACE}";
2070 FOR I=0 TO 9
2080 :PRINT PO(I);"{LEFT}";
2090 NEXT I
2100 FOR I=0 TO 9:IF PO(I)=I THEN NEXT I:RETURN
2110 PRINT:INPUT "{2DOWN}AB WELCHER POSITION DREHEN (0-9) ";AN$
2120 AN=VAL(AN$)
2130 IF AN<0 OR AN>9 THEN GOTO 2110
2140 MI=INT ((9-AN)/2)
2150 FOR I=AN TO AN+MI
2160 :H=PO (I)
2170 :PO (I)=PO(9+AN-I)
2180 :PO (9+AN-I)=H
2190 NEXT I
2200 DG=DG+1
2210 GOTO 2030
3000 REM
3010 REM *** ENDE ***
3020 REM
3030 PRINT:PRINT "{2DOWN}FINITO !"
3040 PRINT "{2DOWN}DAFUER BENOETIGTEN SIE";DG;"DURCHGAENGE."
3050 PRINT
3060 IF DG<11 THEN PRINT "SEHR GUT !":RETURN
3070 IF DG<21 THEN PRINT "NA JA.":RETURN
3080 PRINT "UEBEN SIE MAL NOCH FLEISSIG !":RETURN
```

Programmbeschreibung für Zahlendreher

In Zeile 1030 stoßen Sie auf einen bereits bekannten Variablentyp, das Array. In diesem konkreten Fall wird ein Zahlenarray festgelegt, das neun Elemente aufnehmen soll. Beachten Sie aber bitte eins: Die Numerierung beginnt mit dem Index null. Das heißt, Sie können in einer solchen Feldvariable immer ein Element mehr speichern, als die Zahl beim DIM-Befehl angibt.

Sie können das auch gleich in der nächsten Zeile sehen: Die Schleife, die diese Variable vorbelegt (initialisiert) beginnt bei null und hört bei neun auf. Sollten Sie es mir nicht glauben, so zählen Sie die Elemente nacheinander ab: Sie werden insgesamt auf zehn Einträge kommen.

Der Zweck dieser Schleife ist es, die Liste, die später noch durcheinandergemischt wird, erst einmal in der »Urform« herzustellen. Dabei wird jeder Feldvariablen als Inhalt ihr Index zugewiesen, das heißt also: PO(0)=0, PO(1)=1, PO(2)=2 ... (»PO« übrigens für POsition bzw. PositiOn).

Da diese Schleife nichts Weltbewegendes leistet, konnte sie ruhig in einer Zeile untergebracht werden. Mit der nächsten Schleife werden die Positionen in der Liste zufällig vertauscht. Dazu durchläuft die Schleife alle Indizes des Feldes, um sicher zu gehen, daß auch wirklich jedes Feld berücksichtigt wurde.

Als erstes wird bei jedem Durchgang ein »X« bestimmt; dieses »X« dient als Index für die Variable, deren Inhalt mit dem von PO(I) vertauscht werden soll. Daraufhin werden die Werte dieser beiden Variablen (nämlich PO(X) und PO(I)) ausgetauscht; dazu wird (mal wieder) eine Hilfsvariable »H« benötigt, um keinen Wert während des Tausches zu verlieren. Nach diesen Vorbereitungen kehrt der Interpreter wieder in das Hauptprogramm zurück.

Von dort aus geht es gleich weiter in den Programmteil »SPIEL«. In den ersten vier Zeilen (bis 2060) wird lediglich der Bildschirm beschrieben. Auch die nächsten drei Zeilen sollen etwas auf die Mattscheibe bringen, und zwar die Liste. Sobald Sie das Programm einmal laufen lassen, wird hier zum ersten Mal die (völlig verdrehte) Liste ausgegeben. Wie schon einmal erwähnt, druckt der C64 vor und hinter jeder Zahlenvariable ein [SPACE]. Damit die einzelnen Ziffern aber nur jeweils ein Leerzeichen voneinander getrennt sind, muß eines dieser beiden [SPACE] umgangen werden; und genau dies bewirkt das Zeichen für [CRSR-LEFT] am Ende von 2080.

In Zeile 2100 steht die Abfrage, ob Sie alle Zahlen wieder an die richtige Position bekommen haben. Finden Sie diese Überprüfung ein bißchen kurz? Wir können sie ja einmal gemeinsam durchgehen:

Die Schleife durchläuft alle möglichen Indizes für »PO«. Sollte der Inhalt von PO(I) gleich dem Index sein, ist also an dieser Stelle die richtige Ziffer (siehe auch Initialisierung in 1040). Ist die Bedingung erfüllt, trifft der Computer auf den NEXT-Befehl und überprüft demnach die nächsthöhere Zahl. Das geht so weiter, bis der Index den Wert neun erreicht

hat. Ist auch hier die Bedingung erfüllt, bemerkt der C64, daß die Schleife schon völlig abgearbeitet ist, sucht sich also den nächsten Befehl in dieser Zeile und trifft auf »RETURN«.

Ist aber bei irgendeinem Durchlauf die Bedingung nicht erfüllt, springt der Computer gleich in die nächste Zeile und macht da weiter. Für ihn ist die FOR-NEXT-Schleife zwar noch nicht beendet, aber er wird wohl kein NEXT mehr finden. Denn bevor er wieder auf eines treffen könnte (Zeile 2190), wird die Zählvariable »I« schon wieder verwendet; das heißt, er kann sich ab Zeile 2150 gar nicht mehr daran erinnern, daß er noch eine offene FOR-Schleife hat!

Sind also alle Zahlen an der richtigen Position, kehrt das Programm zurück. Andernfalls werden Sie erst einmal gefragt, ab welcher Position Sie drehen möchten. Ihre Antwort wird in einem String (AN$ wie ANtwort) gespeichert und später weiterverarbeitet. Aber ich möchte noch ein Wort zu dem PRINT-Befehl sagen: Vielleicht denken Sie sich, man könne sich diesen einen Befehl sparen, und dafür in den INPUT-Teil noch ein [CRSR-DOWN] mehr einfügen; prinzipiell bewirkt das dasselbe. Aber wenn Sie sich die letzte Ausgabe in 2080 ansehen, stellen Sie fest, daß sie mit einem Strichpunkt abgeschlossen wird. Und das bedeutet, daß der Cursor an dieser Stelle »stehen« bleibt. Wenn Sie jetzt die nächste Ausgabe mit drei [CRSR-DOWN]s beginnen, wird der Text zwar um drei Zeilen nach unten geschoben, aber er beginnt erst an der Spalte, an der die vorherige Ausgabe (Zeile 2080) aufgehört hat; der PRINT-Befehl dagegen bewirkt, daß der Cursor nicht nur eine Zeile tiefer, sondern auch noch an deren Anfang gesetzt wird.

Probieren Sie es aus: Ersetzen Sie die PRINT-Anweisung durch ein [CRSR-DOWN] innerhalb des INPUT-Befehls. Und Sie werden sofort den Unterschied bemerken.

In Zeile 2120 wird Ihre Eingabe einer Zahlenvariable zugeordnet, damit man mit der eingegebenen Ziffer weiterrechnen kann. Anschließend wird die Eingabe noch geprüft. Sollte sie außerhalb des gültigen Bereichs liegen, verzweigt das Programm wieder zur Eingabe (Zeile 2110).

Mit dem Programmteil »9-AN« berechnet der Computer in Zeile 2140, wie viele Positionen zwischen Ihrer Eingabe und dem Ende liegen. Diese Zahl wird noch halbiert. Wenn man zu Ihrer Antwort »MI« addiert (oder von 9 »MI« subtrahiert), kommt man genau zur MItte zwischen »AN« und 9.

Das macht sich auch die Schleife ab Zeile 2150 zu Nutze. Sie zählt von Ihrer Eingabe bis zur Mitte und vertauscht dabei das aktuelle Element PO(I) mit dem auf der anderen Seite PO(9+AN-I). Auch für diesen Tausch benötigt man wieder eine Hilfsvariable »H«. Sobald dieser Tausch vollendet wurde, wird noch eine Variable DurchGang (DG) um eins erhöht, und ein Sprung nach 2030 leitet die nächste Runde ein. Den Rest spare ich mir! Den müssen Sie einfach beherrschen!

7 Faßfarben

Einführung

Das Prinzip dieses Spiels ist schnell erklärt. Sie sehen drei Spalten mit jeweils sechs Zeilen vor sich auf dem Bildschirm. Jede Zeile hat eine andere Farbe, wobei aber die drei Säulen untereinander gleich sind. Das unterste rechte Feld ist leer. Man kann nun dieses Feld über die drei Säulen hin- und herbewegen.

Möchte man dieses freie Feld nach links bewegen, so bedeutet das, daß das Feld, das momentan links vom freien ist, mit diesem vertauscht wird. Für rechts gilt natürlich dasselbe, nur gerade andersherum. Zu Beginn des Spiels sehen Sie die drei Säulen im Ausgangszustand. So sollten sie auch am Ende wieder aussehen...

Dann werden die Farben vor Ihren Augen gemischt. Ihre Aufgabe ist es nun, die drei Spalten wieder in ihre Ausgangsform zu bringen. Dabei können Sie die Spalten rollen, das heißt, daß jedes einzelne Feld um eins nach oben rutscht, während das oberste Feld wieder ganz unten erscheint.

Zusätzlich können Sie eben das freie Feld einsetzen, um Farben von einer Säule zu einer anderen zu transportieren. Sollten beispielsweise in Spalte 1 zwei von drei schwarzen Feldern liegen, müssen Sie dieses freie Feld verwenden, um ein schwarzes Feld zu Säule 2 bzw. 3 zu transportieren.

Diese Ausführungen klingen im Moment komplizierter als sie in Wirklichkeit sind. Sobald Sie sich an den Computer setzen und die Farbsäulen vor sich sehen, wird Ihnen das System klar werden. Sie sehen am Bildschirm auch noch die Erklärungen, wie Sie die Balken rollen bzw. das freie Feld bewegen.

Listing zu Faßfarben

```
10 REM *****************
20 REM ***           ***
30 REM *** FASSFARBEN ***
40 REM ***           ***
50 REM *****************
60 REM
70 GOSUB 1000
80 GOSUB 2000
90 GOSUB 3000
100 END
1000 REM
1010 REM *** VORBEREITUNGEN ***
1020 REM
1030 DIM FE (3,6)
1040 DIM FA$(6)
1050 FOR I=0 TO 6:READ FA$(I):NEXT I
1060 DATA "{GREY 2}", "{BLACK}", "{RED}", "{LIG.GREEN}", "{YELLOW}", "{C
     YAN}", "{WHITE}"
1070 FOR I=1 TO 3
1080 :FOR J=1 TO 6
1090 : FE (I,J)=J
1100 :NEXT J
1110 NEXT I
1120 FE(3,6)=0
1130 LX=3:LY=6
1140 PRINT "{CLR}"
1150 FOR I=0 TO 18 STEP 3
1160 :POKE 211,11:POKE 214,I:SYS 58640
1170 :PRINT "{RVSON}DDDDDDDDDDDDDDDDDDDD"
1180 NEXT I
1190 FOR I=1 TO 3:GOSUB 1800:NEXT I
1200 POKE 211,0:POKE 214,20:SYS 58640
1210 FOR I=1 TO 3
1220 :PRINT "{RVOFF,35SPACE}"
1230 NEXT I
1240 POKE 211,5:POKE 214,20:SYS 58640
1245 PRINT "{BLACK}<<< DAS IST DAS FERTIGE MUSTER >>>"
1250 FOR PA=1 TO 1500:NEXT PA
1260 PRINT "{UP,RVOFF,27SPACE}"
1270 PRINT "{UP,BLACK,3SPACE}<<< ICH MISCHE JETZT DAS BRETT >>>{3SPACE}"
1280 FOR N=1 TO INT (RND(1)*10)+20
1290 :IF LX=1 OR LX=3 THEN RX=2:GOTO 1310
1300 :RX=INT (RND(1)*2)*2+1
1310 :FOR L=1 TO INT (RND(1)*5)+1
1320 : FOR M=1 TO 6
1330 :  FE(RX,M-1)=FE(RX,M)
1340 : NEXT M
1350 : FE(RX,6)=FE(RX,0):I=RX
1360 :NEXT L:GOSUB 1800
1370 :FE(LX,LY)=FE(RX,LY):FE(RX,LY)=0
1380 :I=LX:J=LY:GOSUB 1840
1390 :LX=RX
1400 :I=LX:J=LY:GOSUB 1840
1420 NEXT N
1430 RETURN
1800 FOR J=1 TO 6
1810 :GOSUB 1840
1820 NEXT J
1830 RETURN
1840 PRINT FA$(FE(I,J));:IF J=0 THEN J=2
1850 POKE 211,(I+1)*6:POKE 214,J*3-2:SYS 58640
1860 PRINT "{RVSON,5SPACE,RVOFF,DOWN,5LEFT}";
```

```
1870 PRINT "<RVSON,5SPACE>";
1880 RETURN
2000 REM
2010 REM *** SPIEL ***
2020 REM
2025 POKE 211,0:POKE 214,20:SYS 58640
2030 PRINT "<RVOFF,39SPACE>"
2040 PRINT "<UP,BLACK,6SPACE,RVSON,2SPACE>1 = ROLLEN DER SPALTE 1<3SPACE>'
2050 PRINT TAB(6);"<RVSON,2SPACE>2 = ROLLEN DER SPALTE 2<3SPACE>"
2060 PRINT TAB(6);"<RVSON,2SPACE>3 = ROLLEN DER SPALTE 3<3SPACE>"
2070 PRINT TAB(6);"<RVSON,SPACE>F1 = N.LINKS F3 = N.RECHTS ";
2080 GET A$:IF A$="" THEN 2080
2090 IF A$="1" OR A$="2" OR A$="3" THEN GOTO 2240
2100 IF A$="<F1>" THEN GOTO 2130
2110 IF A$="<F3>" THEN GOTO 2190
2120 GOTO 2080
2130 RX=LX-1
2140 IF RX=0 THEN GOTO 2080
2150 FE(LX,LY)=FE(RX,LY):FE(RX,LY)=0
2160 I=LX:J=LY:GOSUB 1840
2170 LX=RX:I=LX:J=LY:GOSUB 1840
2180 GOTO 2300
2190 RX=LX+1:IF RX=4 THEN GOTO 2080
2200 FE(LX,LY)=FE(RX,LY):FE(RX,LY)=0
2210 I=LX:J=LY:GOSUB 1840
2220 LX=RX:I=LX:J=LY:GOSUB 1840
2230 GOTO 2300
2240 RX=VAL(A$)
2250 FOR M=1 TO 6:FE(RX,M-1)=FE(RX,M):NEXT M
2260 FE(RX,6)=FE(RX,0):I=RX:GOSUB 1800
2270 IF RX=LX THEN LY=LY-1:IF LY=0 THEN LY=6
2300 FE(LX,LY)=6
2310 FOR I=1 TO 6
2320 :IF FE(1,I)=FE(2,I) AND FE(2,I)=FE(3,I) THEN NEXT I:RETURN
2330 FA(LX,LY)=0
2340 GOTO 2080
3000 REM
3010 REM **** ENDE ****
3020 REM
3030 PRINT "<CLR,2DOWN>VIELEN DANK FUER DIESES SPIEL !"
3040 PRINT "<DOWN>BEEHREN SIE UNS BALD WIEDER."
3050 PRINT "AUF WIEDERSEHEN."
3060 RETURN
```

■ Programmbeschreibung zu Faßfarben

Zu Beginn der »Vorbereitungen« werden wieder zwei Array-Variablen vereinbart. Bei dem ersten Feld handelt es sich um ein zweidimensionales Array, da es zwei Indizes hat, mit denen man auf die einzelnen Komponenten zugreifen kann. Sie können sich diese Anordnung wie eine Tabelle auf dem Papier vorstellen (oder auch wie den Bildschirm vor Ihnen). Vertikal sind die Spalten, und horizontal sind die Zeilen angeordnet. Wenn Sie eine Tabelle aufstellen, gehen Sie folgendermaßen vor: Sie erstellen für eine Zeile alle Spalten, dann kommt die nächste Zeile mit all ihren Spalten...

Und so arbeitet man auch mit zweidimensionalen Arrays: Der erste Index stellt die Zeilen dar, während der zweite Index die Spalte beschreibt. Ich möchte Sie hier auf die Zeilen 1070 bis 1110 hinweisen. Dort können Sie hervorragend sehen, wie man am besten mit

solchen Feldern arbeitet. Dafür werden zwei Schleifen ineinander geschachtelt. Sie müssen natürlich darauf achten, den beiden Schleifenvariablen (hier »I« und »J«) unterschiedliche Namen zu geben! Die äußere Schleife läuft von eins bis drei, bedient also die Zeilen bzw. den ersten Index, während die innere Schleife die Spalten versorgt; gleichzeitig wird den Spalten auch noch der jeweilige Wert dieser inneren Schleife zugewiesen. So einen ähnlichen Fall hatten wir schon einmal bei dem Programm Zahlendreher.

Die Variable »FA$« wird als eindimensionales String-Array mit sechs Teilvariablen angelegt. In einer Schleife in Zeile 1050 wird jedem dieser Felder eine Farbe zugewiesen, die durch eine READ-Anweisung aus der Zeile 1060 gelesen werden.

Beachten Sie bitte den Unterschied: Anscheinend bleiben die FElder FE(1,0), FE (2,0) und FE (3,0) ungenutzt, während bei dem Feld »FA$« alle Komponenten von null bis sechs genutzt werden (darauf habe ich Sie im vorhergehenden Kapitel hingewiesen). Doch dieses Feld mit dem Index null macht sich das Programm später noch zu nutze. Tatsache ist allerdings, daß die Felder FE (0,0) ... FE (0,6) ungenutzt bleiben, doch das soll uns im Moment nicht stören. Ich möchte Ihnen nur deutlich machen, daß man durch diese Leichtsinnigkeit viel Speicherplatz verschenken kann, da für jede einzelne Komponente wertvolles RAM freigehalten wird.

Nachdem (fast) alle Felder belegt worden sind, wird ein ganz spezielles in Zeile 1120 wieder auf null gesetzt. An der Stelle, die durch diese Variable repräsentiert wird, ist eine freie Stelle.

Wenn Sie sich vielleicht an den Bildschirmaufbau dieses Spiels erinnern: Er bestand aus drei Säulen zu je sechs Zeilen; erkennen Sie den Zusammenhang zwischen diesem Aufbau und der Variablen »FE«? Ich muß allerdings zugeben, daß in diesem Fall Zeilen und Spalten um 90 Grad gedreht wurden, aber es sollte dennoch möglich sein, diesen Aufbau nachzuvollziehen (ich meine damit nur, daß die Zeilen mit den Spalten vertauscht wurden und umgekehrt). Dem Computer ist es egal, welcher Index Zeile bzw. Spalte repräsentiert.

Wenn Sie alle Farben in der richtigen Reihenfolge haben wollen, muß das eine freie Feld rechts unten in der dritten Spalte in der sechsten Zeile sein. Und das wird hier festgelegt. Sobald das Programm also bis Zeile 1120 abgearbeitet wurde, ist die Ausgangssituation festgelegt.

Nicht genug damit, daß eine Variable innerhalb von »FE« das freie Feld darstellt; das Programm muß immer wissen, wo sich dieses freie Feld im Moment befindet. Dafür werden zwei Variablen »LX« und »LY« (»L« für Leer) eingeführt. Darin wird festgehalten, welche »Koordinaten« das leere Feld hat (die »Koordinaten« beziehen sich auf die Spalten und Zeilen, so daß »X=3« und »Y=6« die Stelle rechts unten bezeichnen).

Nach diesen Vorbereitungen wird erst einmal das Original-Feld so auf dem Bildschirm ausgegeben. Dafür sind die Zeilen 1140 bis 1250 zuständig. Dabei wird eine Unterroutine ab Zeile 1800 verwendet, die wiederum die eigentliche Ausgaberoutine ab Zeile 1840 aufruft. In diesem Fall sieht die Verschachtelung (von GOSUBs, Sie erinnern sich?) wie folgt aus:

Zeile 70 ruft Zeile 1000 auf

:Von Zeile 1190 aus wird nach Zeile 1800 verzweigt

: Zeile 1800 benutzt die Unterroutine ab Zeile 1840

Auch hier wurden mehrere GOSUBs geschachtelt; keine Sorge, der C64 bringt die Zeilennummern nicht durcheinander, solange Sie jedem GOSUB ein RETURN zuordnen.

Nach einer kurzen PAuse (Variable »PA«) in Zeile 1250 und einer weiteren Bildschirmausgabe wird das Muster gemischt. Die äußerste Schleife (Zeile 1280) wird zwischen 20- und 29mal durchlaufen; in den nächsten beiden Zeilen wird getestet, in welcher Spalte sich das leere Feld befindet. Ist es in der linken oder rechten Spalte, kann es nur in die mittlere verschoben werden; ist es dagegen in der mittleren, kann es entweder nach rechts oder nach links verschoben werden; diese Auswahl erledigt der Zufallsgenerator in 1300. Schauen wir uns diese Zeile etwas näher an: RND(1)*2 produziert Zufallszahlen zwischen null (einschließlich) und zwei (ausschließlich); INT schneidet die Nachkommastellen ab, das heißt, man erhält null oder eins; dieses Ergebnis wird mit zwei multipliziert, also hat man null oder zwei. Die nächste Schleife wird zwischen ein- und viermal durchlaufen, die nächste immer sechsmal. Innerhalb dieser innersten Schleife wird die Säule »RX« (wurde oben bestimmt) um eins nach oben versetzt: FE(RX,M-1)=FE(RX,M).

Hier macht man sich jetzt das Feldelement mit dem Index 0 zu Nutze: Da das Feld nach oben gerollt wird, müßte man die oberste Farbe in einer Hilfsvariablen retten, um sie dann nach dem »Rollen« in der untersten Zeile wieder zu speichern. Da man aber das Feldelement 0 nicht nutzt, kann man es praktisch als Hilfsvariable verwenden: das ist aber um so praktischer, als man keine neue Variable (zum Beispiel »H«) einführen muß, sondern sie gleich mit dem normalen Index ansprechen kann. Ich glaube, wenn Sie versuchen, die Zeilen 1320 bis 1350 nachzuvollziehen, müßte klarwerden, was ich meine. Schließlich wird die »Hilfsvariable« noch in die unterste Zeile wieder übertragen (Zeile 1350).

Diese innerste Schleife wird selbst mehrmals wiederholt (wegen der Schleife »L«); anschließend wird das Ergebnis zwischendurch einmal auf dem Bildschirm ausgegeben. Dadurch können Sie am Monitor schön verfolgen, wie die Ordnung allmählich dem Chaos weicht, wie Ihre Arbeit wächst und wächst ...

Nach dieser Schleife wird noch das freie Feld in die eben verdrehte Spalte verschoben (Zeile 1370). Würde man lediglich alle drei Spalten mehrmals nach oben schieben, hätte das wenig Einfluß auf die Farben. Man muß noch Farben zwischen den Säulen austauschen. Und das darf auch der Computer nur mit dem freien Feld machen. Dadurch wird sichergestellt, daß das vermischte Feld entwirrbar (lösbar) bleibt.

Der Variablen, die eben noch das freie Feld beherbergte, wird der Wert der Stelle zugewiesen, mit der getauscht wird. Da dieser Tausch nur in derselben Zeile erfolgen kann, bleibt bei beiden Variablen das »LY« stehen (FE(LX,LY)=FE(RX,LY)). Da man von vornherein weiß, welcher Wert der Variablen FE(RX,LY) zugeordnet wird (nämlich null), muß man keine Hilfsvariable einführen. Nach diesem Tausch werden den Variablen »I« und »J« die

»alten« Koordinaten des leeren Feldes zugewiesen, und die Ausgabe eines Feldes wird angesprungen (ab Zeile 1840). Anschließend werden noch die Koordinaten auf den neuesten Stand gebracht (Zeile 1390; »LY« bleibt ja gleich!), und wieder wird die Ausgaberoutine angesprungen, damit die Bildschirmausgabe »up to date« ist.

Sobald die äußerste Schleife abgearbeitet ist, kehrt der C64 wieder in das Hauptprogramm zurück. Ab der Zeile 1840 steht die Ausgabe für ein einzelnes Feld. In den Variablen »I« und »J« müssen die »Koordinaten« angegeben sein; die Routine ab 1800 benützt diese Unterroutine nur, um alle Felder ausgeben zu lassen; dafür werden am praktischsten zwei Schleifen benutzt, die als Variablen gleich »I« und »J« verwenden.

Damit dürfte der Teil »Vorbereitungen« zur Genüge besprochen worden sein. Dann geht es jetzt gleich weiter: Mit den ersten paar Befehlen wird (mal wieder) die Bildschirmausgabe gesteuert. So soll am unteren Bildschirmrand die Bemerkung » <<< ICH MISCHE JETZT DAS BRETT >>> « gelöscht und eine Kurzanleitung ausgegeben werden.

Sollte bei der Eingabe in Zeile 2080 eine eins, zwei oder drei eingegeben worden sein, verzweigt der Interpreter zu Zeile 2240, um dort jeweils das »Rollen« der einzelnen Spalten zu veranlassen.

In den nächsten zwei Zeilen können Sie sehen, wie einfach man die F-Tasten (die dunkelgrauen Tasten am rechten Rand des Gehäuses, bzw. am rechten oberen Rand beim C128) abfragen kann: einfach ein Anführungszeichen setzen und dann die gewünschte Taste drücken, also in Zeile 2100 die F1-Taste bzw. in Zeile 2110 die F3-Taste. Je nachdem wird auch verzweigt; bei allen anderen Eingaben kehrt das Programm wieder zu Zeile 2080 zurück, um auf eine sinnvolle Eingabe zu warten.

Ab Zeile 2130 wird der Fall behandelt, daß F1 gedrückt wurde. Das heißt, Sie wollen das leere Feld nach links verschieben, folglich muß die Spalte die Koordinate »LX-1« haben; und dieser Wert wird »RX« zugewiesen. Sollte RX=0 sein, das heißt, das leere Feld befand sich schon ganz links, wird wieder zu Zeile 2080 verzweigt.

Andernfalls werden die Inhalte der beiden Felder RX/LY und LX/LY miteinander vertauscht und anschließend jeweils ausgegeben (Zeilen 2160 und 2170). Für die F3-Taste gilt prinzipiell das Gleiche, nur jeweils spiegelbildlich; das heißt, »RX« wird der Wert LX+1 zugewiesen; entsprechend muß die Abfragebedingung in Zeile 2190 heißen: IF RX=4 THEN ...

Sollten Sie aber 1, 2 oder 3 eingegeben haben, wird das Ganze etwas umfangreicher. In dem Fall kann man ganz einfach der Variablen »RX« den Zahlenwert Ihrer Eingabe mit »VAL (A$)« zuordnen (»A$« wie Antwort).

Dann läßt sich die betreffende Reihe mit einer FOR-NEXT-Schleife einfach um eins weiterdrehen; das Spielchen mit dem Index 0 hatten wir ja schon einmal. Die Zeilen 2250 und 2260 sollten also klar sein. In Zeile 2270 wird noch geprüft, ob sich das freie Feld innerhalb dieser einen Spalte befand. Ist das der Fall, muß noch die Y-Koordinate »LY«

aktualisiert werden (LY=LY−1). Ist »LY« daraufhin null (was das freie Feld also in der obersten Zeile darstellt), wird »LY« auf sechs gesetzt (jetzt ist es dann in der untersten!).

Ab Zeile 2300 kommt die Überprüfung, ob sich bereits alle Felder in der richtigen Reihenfolge befinden. Dazu muß zunächst der Inhalt des freien Feldes auf sechs gesetzt werden. Sie wissen doch: Damit das Programm das Ende erkennt, muß das freie Feld in der untersten Spalte sein, und alle Zeilen müssen dieselbe Farbe aufweisen. Daher ist es am einfachsten, dem leeren Feld die Farbe zuzuweisen, die es in der untersten Zeile haben sollte, und dann alle Zeilen zu überprüfen.

Ist diese Bedingung für alle Zeilen erfüllt, dann springt das Programm zurück (mit RETURN). Findet der C64 aber auch nur eine Zeile, bei der die Farben nicht übereinstimmen, arbeitet er die Zeile nach dem IF-Befehl ab, und die sagt ihm, den Inhalt des leeren Feldes wieder auf null zu setzen, und wieder auf eine Antwort zu warten − Sie müssen also weiterspielen.

Na, wenn Ihnen das Ende jetzt noch Schwierigkeiten macht ...

8 Fünf-in-einer-Reihe

Einführung

Vielleicht kennen Sie dieses Spiel als »Drei in einer Reihe«. Ich habe die Regeln ein klein wenig umgeformt und daraus eben »Fünf-in-einer-Reihe« gemacht. Sie müssen zum Laden dieses Programmes den Namen »Fuenfer« als File-Titel verwenden; die Floppy 1541 akzeptiert nur Namen bis zu einer Länge von 16 Zeichen. Daher mußte ich den Namen auf »Fuenfer« abändern, da ich auf eine Abkürzung verzichten wollte.

Nun aber zum Spiel: Wie Sie sich leicht vorstellen können, geht es darum, fünf Spielsteine in eine Reihe zu bekommen. Ihre Steine können horizontal oder vertikal liegen, nur diagonal dürfen sie nicht angeordnet sein. Das heißt, sie dürfen so angeordnet sein, nur erkennt der Computer keine solche Reihe – insofern bringt dieses System nicht allzuviel.

Der Computer ist nach Ihrem Zug dran. Dabei setzt er seine Spielsteine willkürlich (sprich zufällig). Dadurch kann er Ihre fast vollständige Reihe im letzten Moment noch vermasseln. Im Programm wurde bewußt darauf verzichtet, eine spezielle Taktik zu programmieren. Für den Spielreiz wäre dies sicher förderlich, aber für ein Basic-Einführungsbuch nicht besonders sinnvoll.

Ihre Steine setzen Sie einfach durch die Angabe der Reihe, in der Sie Ihren Spielstein gerne untergebracht haben möchten. Der Computer »wirft« Ihre Figur dann an besagter Spalte oben ein, und diese fällt nach unten durch bis zur letzten freien Zeile.

Gewonnen hat, wer als erster fünf Spielsteine nebeneinander oder übereinander angeordnet hat.

Listing zu Fünf-in-einer-Reihe

```
10 REM ****************************
20 REM ***                      ***
30 REM *** FUENF-IN-EINER-REIHE ***
40 REM ***                      ***
50 REM ****************************
60 REM
70 GOSUB 1000
80 GOSUB 2000
90 GOSUB 3000
100 END
1000 REM
1010 REM *** VORBEREITUNGEN ***
1020 REM
1030 PRINT "{CLR}";TAB(6);"{RVSON,2SPACE}1{3SPACE}2{3SPACE}3{3SPACE}4{3S
     PACE}5{3SPACE}6{3SPACE}7{2SPACE}"
1040 FOR I=1 TO 7
1050 :FOR J=1 TO 2
1060 : PRINT TAB(6);"{RVSON,SPACE,RVOFF,3SPACE,RVSON,SPACE,RVOFF,3SPACE,
     RVSON,SPACE,RVOFF,3SPACE}";
1070 : PRINT "{RVSON,SPACE,RVOFF,3SPACE,RVSON,SPACE,RVOFF,3SPACE,RVSON,S
     PACE,RVOFF,3SPACE}";
1080 : PRINT "{RVSON,SPACE,RVOFF,3SPACE,RVSON,SPACE,RVOFF}"
1090 :NEXT J
1100 :PRINT TAB(6);"{RVSON,29SPACE}"
1110 NEXT I
1120 PRINT "{UP}";TAB(6);"{RVSON,2SPACE}1{3SPACE}2{3SPACE}3{3SPACE}4{3SP
     ACE}5{3SPACE}6{3SPACE}7{2SPACE}"
1130 Z$(1)="OOO"
1140 Z$(2)="***"
1150 C(1)=15
1160 C(2)=42
1170 DEF FN X(X)=(X-1)*4+7
1180 DEF FN Y(X)=(X-1)*3+1
1190 DEF FN BS(X)=PEEK(1024+X)
1200 POKE 53280,14:POKE 53281,6
1210 RETURN
2000 REM
2010 REM *** SPIEL ***
2020 REM
2030 POKE 211,0:POKE 214,22:SYS 58640
2050 PRINT "{40SPACE}"
2070 P=1
2080 INPUT "{LIG.BLUE,UP}IHR ZUG (SPALTE 1-7) ";AN
2090 IF AN>0 AND AN<8 THEN GOTO 2130
2110 PRINT "BITTE EINE ZAHL ZWISCHEN 1 UND 7{2UP}"
2120 GOTO 2080
2130 SX=FN X(AN)
2140 IF FN BS(SX+40)=32 THEN GOTO 2180
2160 PRINT "DIESE ZEILE IST SCHON VOLL !{2UP}"
2170 GOTO 2080
2180 PRINT "{34SPACE}";
2185 FOR J=1 TO 7
2190 :PRINT MID$("{BLACK,WHITE}",P,1);
2200 :SY=FN Y(J)
2220 :POKE 211,SX:POKE 214,SY:SYS 58640
2230 :PRINT Z$(P);"{DOWN,3LEFT}";
2240 :PRINT Z$(P)
2250 :IF J=7 THEN GOTO 2320
2260 :LX=FN Y(J+1)
2270 :IF FN BS(SX+LX*40)<>32 THEN J=7:GOTO 2320
2290 :POKE 211,SX:POKE 214,SY:SYS 58640
```

```
2300 :PRINT "{3SPACE,DOWN,3LEFT}";
2310 :PRINT "{3SPACE}"
2320 NEXT J
2330 FOR I=1 TO 7
2340 :FOR J=1 TO 3
2350 : FOR K=J TO J+4
2360 :  IX=FN X(I)
2370 :  KX=FN Y(K)
2380 :  IF FN BS(IX+KX*40)<>C(P) THEN K=J+4:NEXT K:GOTO 2410
2390 : NEXT K
2400 : RETURN
2410 :NEXT J
2420 NEXT I
2430 FOR J=1 TO 7
2440 :FOR I=1 TO 3
2450 : FOR K=I TO I+4
2460 :  KX=FN X(K)
2470 :  JX=FN Y(J)
2480 :  IF FN BS(KX+JX*40)<>C(P) THEN K=I+4:NEXT K:GOTO 2510
2490 : NEXT K
2500 : RETURN
2510 :NEXT I
2520 NEXT J
2530 IF P=2 THEN P=1:GOTO 2030
2540 POKE 211,0:POKE 214,23:SYS 58640
2550 PRINT "{40SPACE}";
2570 PRINT "{UP,LIG.BLUE}MEIN ZUG ..."
2580 P=2
2590 AN=INT (RND(1)*7)+1
2600 SX=FN X(AN)
2610 IF FN BS(SX+40)<>32 THEN GOTO 2590
2620 GOTO 2180
3000 REM
3010 REM *** ENDE ***
3020 REM
3030 POKE 211,0:POKE 214,22:SYS 58640
3040 PRINT "{40SPACE}"
3050 PRINT "{40SPACE}"
3060 PRINT"{3UP,LIG.BLUE}DAS SPIEL IST BEENDET !!!"
3070 PRINT "GEWONNEN ";
3080 IF P=1 THEN PRINT "HABEN SIE !"
3090 IF P=2 THEN PRINT "HABE ICH !"
3100 RETURN
```

■ Programmbeschreibung für Fünf-in-einer-Reihe

Bis zur Zeile 1120 wird lediglich das Grundgerüst für den Bildschirmaufbau gelegt. Dazu gibt das Programm erst einmal auf den leeren Bildschirm die Nummern der einzelnen Spalten aus (Zeile 1030).

Danach sind zwei ineinander verschachtelte Schleifen dafür zuständig, die richtige Spalte bzw. Zeile an den richtigen Platz zu bringen. Zunächst wird der Cursor dafür in die sechste Spalte des Bildschirms gesetzt (Zeile 1060). In den folgenden PRINT-Anweisungen wechseln sich immer drei [SPACE]s mit einem Stück Balken ab. Dieser wird bekanntlich durch ein inverses [SPACE] erzeugt, sprich: [CTRL]+[9] [SPACE] [CTRL]+[0]. Mit dieser Zeichenfolge werden also die Zeilen 1060 bis 1080 erstellt.

Da die letzte der drei Zeilen (Zeile 1080) nicht mit einem Strichpunkt abgeschlossen wird, steht der Cursor zu Beginn der nächsten Ausgabe am Anfang der Zeile. Und dort werden die eben besprochenen Zeilen noch einmal ausgegeben, da die Schleife ab Zeile 1050 zweimal durchlaufen wird.

Wurde also eine »freie Zeile« ausgegeben (innerhalb der J-Schleife), muß vor der nächsten Zeile eine Unterteilung erfolgen. Und das geschieht in Zeile 1100, bevor die äußere Schleife noch einmal »durchackert« wird. Den genauen Vorgang zu beschreiben, ist etwas schwerfällig; leichter ist es, wenn Sie das Programm einfach einmal laufen lassen, sich den Bildschirmaufbau anschauen und das Listing damit vergleichen.

Nach der (erfolgreichen) Ausgabe dieses Spielfeldes, wird am unteren Rand noch einmal die Nummer der jeweiligen Spalte angegeben.

Anschließend werden die Zeichen festgelegt, die für die beiden Spieler ausgegeben werden sollen. Für Spieler 1 wird als Zeichen dreimal das »O« ausgegeben, während Spieler 2 durch drei Sternchen repräsentiert wird. Diese beiden Zeichen können Sie gerne verändern. Doch müssen Sie eines beachten. In den nächsten zwei Zeilen (1150 und 1160) wird zwei Variablen der Wert zugewiesen, der für die entsprechenden Zeichen steht.

Und zwar muß das Programm überprüfen können, ob Sie (oder der Computer) bereits fünf Steinchen in einer Reihe haben. Dazu muß es die internen Bildschirmcodes miteinander vergleichen. Ich habe bereits gesagt, daß Sie mit POKE 1024,1 ein »A« auf den Bildschirm bringen können. Um eben diesen Code handelt es sich. Wenn Sie POKE 1024,15 eingeben, werden Sie ein »O« erhalten, mit POKE 1024,42 ein »*«. Verstehen Sie den Zusammenhang?

Wenn nicht, sollten Sie mit den Veränderungen lieber warten, bis Sie sich etwas besser mit den internen Zeichendarstellungen des C64 auskennen, die aber keineswegs schwer sind. Aber vielleicht macht Sie etwas anderes unsicher? Und zwar wird da doch eine Feldvariable benutzt, ohne vorher mit dem DIM-Befehl angemeldet worden zu sein! Das ist richtig. Doch müssen Sie dazu wissen, daß man solche Feldvariable einfach benutzen kann, solange sie nicht mehr als elf Elemente hat. Das gilt aber nur für ein- und zweidimensionale Felder (bis zu XY(10,10)!); bei dreidimensionalen Feldern müssen Sie *alle* Arrays mit »DIM« deklarieren!

Zum Abschluß der Vorbereitungen werden noch drei Funktionen definiert. Und zwar rechnet die erste Funktion namens »X« die Bildschirmspalte aus, wenn Sie die Nummer der »Spielspalte« eingeben. Wenn das Programm Ihren Spielstein »durchfallen« läßt, berechnet »Y« immer die eigentliche Bildschirmkoordinate dazu.

Abschließend liefert »BS« noch den Code des Zeichens, das am Bildschirm an der Stelle »X« steht (dazu später mehr).

Bevor der Computer sich an die Bearbeitung des Hauptteils machen kann, werden noch die Farben auf die Werte gesetzt, die der Computer auch beim Einschalten anzeigt. Sollten Sie beispielsweise vor dem Programm Fünf-in-einer-Reihe Würmli gespielt haben, könnten Sie

den Bildschirmaufbau wegen der Farbeinstellung nur schlecht erkennen. Daher werden vorsichtshalber die Farben zurückgesetzt.

Im Hauptprogramm wird zunächst die Bildschirmzeile 22 gelöscht (2030 und 2050). Anschließend wird festgelegt, daß Spieler 1 an der Reihe ist; da Spieler im Computer-Fachjargon »player« heißt, wurde auch die Variable dementsprechend benannt.

Anschließend erfolgt Ihre Eingabe. Dazu bewegt der Computer den Cursor wegen des [CRSR-UP] noch um eine Zeile höher, damit der vorgesehene Text in die vorher gelöschte Bildschirmzeile gedruckt wird; ansonsten wäre das Löschen dieser einen Zeile ja überflüssig gewesen.

In Zeile 2090 wird gleich geprüft, ob Ihre Eingabe auch gültig war; wenn ja, geht es weiter. Ansonsten werden Sie in Zeile 2110 aufgefordert, nur Zahlen zwischen eins und sieben einzugeben.

In Zeile 2130 sehen Sie die Anwendung der Funktion »X«. Und zwar wird Ihre ANtwort in die X-Koordinate auf dem Bildschirm umgerechnet. Da ein Eintrag ins Spielfeld aus zwei Zeilen besteht, wird zuvor noch überprüft, ob die Bildschirmzelle, die genau unter dem errechneten Wert liegt, noch frei ist. Das geschieht mit der Funktion »BS«. Zu der vorher berechneten X-Koordinate werden 40 Spalten addiert, das entspricht der Länge einer Bildschirmzeile. Das heißt, man erhält mit »SX+40« dieselbe Spalte, nur eine Zeile tiefer. Ist diese Stelle noch frei (der Code für [SPACE] ist 32, und eine Stelle gilt als frei, wenn ein [SPACE] »dasteht«), kann Ihr Zeichen Z$(1) ausgegeben werden. Sollte irgendetwas anderes da stehen, erscheint die Meldung, daß diese Zeile bereits voll ist (Zeile 2160), und Sie müssen noch einmal einen Wert eingeben.

Ab Zeile 2185 kommt erst einmal eine lange Schleife. Sie erstreckt sich bis zu Zeile 2320 und hat allein die Funktion, Ihr Zeichen (sagen wir Spielstein) die ganze Spalte soweit wie möglich hindurchfallen zu lassen. Also so lange eine Zeile weiter nach unten zu schieben, wie die Zeilen darunter frei sind. Dazu wird keinerlei Variable verwendet, in der eventuell die belegten Zeilen und Spalten gespeichert würden. Sondern das Programm arbeitet allein mit dem Bildschirmspeicher, in dem alle notwendigen Daten zur Verfügung stehen. Man kann zwar nicht so bequem wie mit Variablen arbeiten, aber man braucht Daten auch nicht zweimal anzulegen, so lange man recht einfach auf sie zugreifen kann. Doch jetzt zur Schleife. Da das Programm versucht, Ihren Spielstein so tief wie möglich zu plazieren, beginnt es mit der Überprüfung oben und arbeitet sich nach unten durch. Dabei kann man den Spielstein immer gleich anzeigen lassen, falls eine Zeile mal wieder als frei registriert wurde.

In Zeile 2190 werden Sie erst einmal mit einem neuen Befehl konfrontiert. Er heißt MID$. Wenn Sie einen String haben, können Sie mit dieser Funktion gezielt Buchstaben aus dem String »herausholen«. Dazu erst einmal ein Beispiel:

```
PRINT MID$("GRUESS GOTT",8,4)#
```

gibt Ihnen auf dem Bildschirm den Text »GOTT« aus. Und zwar gibt die erste Zahl an, ab welcher Stelle in der Zeichenkette Sie Buchstaben herausholen möchten, und die zweite

Zahl, wie viele Buchstaben es denn sein sollen. Das heißt also für unser Beispiel, ab der achten Stelle waren vier Buchstaben gesucht. Und das ergab das Wort »GOTT«.

Um Mißverständnissen vorzubeugen: »Herausholen« meint in dem Fall nicht, daß der ursprüngliche String die Buchstaben nicht mehr hat. Sie werden kopiert und dann ausgegeben. Ein weiteres Beispiel soll das verdeutlichen:

```
A$="GRUESS GOTT"
B$=MID$(A$,8,4)
PRINT A$, B$
```

Die Bildschirmausgabe sähe in dem Fall so aus:

```
GRUESS GOTT         GOTT
```

Das Komma hinter »A$« läßt die nächste Ausgabe (hier »B$«) bei der nächsten Zehnerspalte beginnen (also bei der 10ten, 20sten, 30sten oder 40sten (nächste Zeile erste) Spalte). In unserem Fall soll aus dem String [CTRL]+[1] [CTRL]+[2] (für die Farben Schwarz und Weiß) ein Buchstabe (also eine Farbe) ausgegeben werden. Welche Farbe das ist, bestimmt die Variable »P«, die enthält, welcher Spieler gerade dran ist. Diese Zeile ist dafür verantwortlich, daß die Zeichen von Spieler eins und zwei in unterschiedlichen Farben dargestellt werden.

In Zeile 2200 wird die Y-Koordinate für den Bildschirm berechnet, die der aktuellen Zeile (also dem Inhalt von »J«) entspricht. Die X-Koordinate wurde schon früher berechnet (Zeile 2130), so daß der Cursor schon an die richtige Stelle gesetzt werden kann (Zeile 2220). Anschließend wird gleich das Zeichen für den jeweiligen Spieler ausgegeben, daher Z$(P). Da das Zeichen aber zweimal untereinander ausgegeben werden soll, muß der Cursor nach dem ersten Mal eine Zeile tiefer und drei Zeichen nach links gesetzt werden. Erst dann kann die zweite Ausgabe erfolgen.

Sollte das Zeichen bereits in der untersten Zeile angekommen sein, dann wäre J=7 und somit die Bedingung für den Sprung in 2250 erfüllt. Der Computer spränge zu Zeile 2320, wo er auf den NEXT-Befehl trifft; da »J« aber bereits die obere Schleifengrenze erreicht hat (siehe Zeile 2185), sucht der Computer den nächsten Befehl, den er in 2330 fände.

Ist die Bedingung (Zeile 2250) aber nicht erfüllt, muß getestet werden, ob die Zeile unterhalb noch frei ist. Dazu wird der Variablen »LX« der Wert zugewiesen, der der nächst unteren Zeile entspricht, nämlich J+1; dieser Wert wird jedoch gleich der Funktion »Y« übergeben, so daß »LY« schließlich die Y-Koordinate für die untere Zeile enthält. Ist die Bildschirmstelle an diesem Platz nicht frei (der Inhalt ungleich 32), darf der Spielstein nicht weiter versetzt werden; darum wird »J« »manuell« auf sieben gesetzt und zum NEXT-Befehl verzweigt (siehe oben).

Vergleichen Sie bitte das Argument für »BS« in Zeile 2270 mit dem in Zeile 2140. Sehen Sie den Unterschied? Der läßt sich ganz einfach erklären: in 2140 wurde lediglich die erste Zeile überprüft. Da es hier jedoch um die zweite, dritte ... Zeile gehen kann, muß man die 40. Spalte noch mit der jeweiligen Zeilennummer multiplizieren, um an der richtigen Bildschirmstelle zu landen.

Sind also alle Bedingungen für ein Landen in der nächsttieferen Zeile erfüllt, muß die aktuelle Zeile noch gelöscht werden (sonst wäre die erste Spalte nach einem Steinchen bereits voll, wenn man die oberen Zeilen nicht wieder löschen würde!). Dazu wird der Cursor noch einmal an dieselbe Stelle gesetzt wie in Zeile 2220, und anschließend Ihre Zeichen mit [SPACE]s überschrieben, sprich gelöscht.

Zum Schluß steht noch der NEXT-Befehl. Wurde zuletzt Ihr Zeichen gelöscht, so bedeutet das, daß »J« noch nicht sieben war (da sonst Abfrage in 2250 positiv gewesen wäre), der Computer wegen des NEXT zu Zeile 2190 springt, »J« jetzt für die nächst tiefere Zeile steht, und dort Ihr Zeichen neu ausgegeben wird ... (siehe oben).

Diese Ausführungen waren sehr trocken und theoretisch. Sollten Sie noch Probleme dabei haben, das Programm nachzuvollziehen, schlage ich Ihnen vor, sich mit dem Listing an den Computer zu setzen, das Programm laufen zu lassen und dabei das Listing zu vergleichen. Dabei sollten alle Probleme gelöst werden.

Jetzt kommen zwar noch zwei weitere Mammut-Schleifen, aber ich hoffe, daß die Erläuterungen dazu kürzer ausfallen können. Diese beiden Schleifen überprüfen nur, ob Sie (oder Ihr Gegenspieler, der Computer) bereits fünf Steine nebeneinander oder übereinander gebracht haben. Dabei ist die Schleife von Zeile 2330 bis 2420 für die vertikale, und Schleife 2 von 2430 bis 2520 für die horizontale Überprüfung verantwortlich.

In der ersten Schleife ist »I« die Zählvariable. Sie zählt alle Spalten nacheinander ab. Dazwischen wird eine Schleife eingelagert, die mit »J« die ersten drei Zeilen überprüft. Und schließlich ist da noch die innerste Schleife mit K, die jeweils von der aktuellen Zeile ab die nächsten vier Zeilen überprüft. Dadurch wird gewährleistet, daß immer fünf Zeilen getestet werden, die auch wirklich nebeneinander liegen. Wird »J« um eins erhöht, bedeutet das, daß mit »K« die nächste Fünfergruppe Zeilen überprüft wird.

Doch jetzt zur eigentlichen Überprüfung: In »IX« und »KX« wird die aktuelle Bildschirmposition gespeichert, die mit den beiden bekannten Funktionen »X« und »Y« zuvor berechnet wurde (Zeilen 2360 und 2370). In Zeile 2380 wird der Code an der Stelle »IX/KX« endgültig überprüft. Ist er gleich dem Code des Spielers, der gerade an der Reihe ist (C(P)), so muß weitergeprüft werden, sprich »NEXT K« wird bearbeitet. Ist der Code ungleich dem Code des Spielers, wird »K« auf die Abbruchbedingung (J+4) gesetzt, ein NEXT-Befehl (praktisch ohne Wirkung) ausgeführt und anschließend zu Zeile 2410 verzweigt (entspricht der nächsten Fünfergruppe).

Wurde der NEXT-Befehl in Zeile 2390 ausgeführt, wird weitergeprüft. Sind alle fünf Steine gleich (und nur dann), kommt der Computer zu dem RETURN in Zeile 2400, da sonst die Befehle in 2380 ausgeführt würden.

Durch das »NEXT J« wird die überprüft Fünfergruppe um eins nach unten geschoben, und durch das anschließende »NEXT I« kommt die nächste Spalte dran. Sollte der Computer bei »I=7« noch auf das »NEXT I« in Zeile 2420 stoßen, bedeutet das, daß der Computer keine vertikale Fünfergruppe finden konnte. In diesem Fall geht das Programm nahtlos in die horizontale Überprüfung über.

Vom prinzipiellen Aufbau her gleicht diese Schleife der ersten wie ein Ei dem anderen. Aber es gibt doch kleine Unterschiede (klein, aber fein!). Mit der äußersten Schleife werden jetzt nicht die Spalten, sondern die Zeilen nacheinander abgetastet. Und die nächsten zwei sind für die Überprüfung der Spalten zuständig. Sie müssen natürlich darauf achten, den Funktionen »X« und »Y« die richtigen Variablen mitzugeben, und dann auch die Variablen in der richtigen Reihenfolge für »BS« (wie BildSchirm) einzusetzen (also KX+JX*40 und nicht JX+KX*40).

Aber ansonsten überlasse ich Sie jetzt mal sich selbst! In Zeile 2530 wird abgefragt, ob gerade Spieler 2 (Computer) an der Reihe war. Ist das der Fall, wird die Variable »P« wieder auf eins gesetzt, und das Programm verzweigt an den Anfang.

Andernfalls beginnt hier der (kurze) Programmteil, der den Computerzug ausführt. Dabei macht sich natürlich bemerkbar, daß der C64 nur zufällig zieht, also keine Bewertung einführt, ob die Lage für ihn brenzlig oder günstig ist. Er kann Ihnen höchstens zufällig einen Strich durch die Rechnung machen. Für den Computerzug wird eine Zufallszahl zwischen eins und sieben bestimmt (Zeile 2590). Da das Programm wieder die obigen Programmteile für die Anzeige und Überprüfung verwenden soll, müssen die gleichen Variablen wie oben verwendet werden. Das bezieht sich zwar noch nicht auf die Variable »AN«, sie wurde aber wegen der Analogie beibehalten. In der Variablen »SX« dagegen muß der umgerechnete Wert stehen, da die Routinen auf diese Variable zurückgreifen. In 2610 wird lediglich noch überprüft, ob die oberste Zeile in der angewählten Zeile frei ist. Ist sie nämlich bereits belegt, müßte die Zufallsvariable noch einmal belegt werden.

In Zeile 2620 erfolgt schließlich ein Sprung nach Zeile 2180, wo ab sofort die oben besprochenen Routinen zum Einsatz kommen.

9 Buchstabendreher

Einführung

Wie der Titel es schon ankündigt, werden in diesem Spiel Buchstaben verdreht, und Sie müssen sie wieder in die richtige Reihenfolge bringen. Dabei wurde diesmal besonderer Wert auf die Bildschirmausgabe gelegt; Sie könne jetzt also einmal sehen, was man alles mit dem PRINT-Befehl machen kann, wenn man ihn richtig einsetzt.

Nun aber zum Spiel: Auf dem Bildschirm sehen Sie ein Spielfeld, das vier mal vier Buchstaben faßt. Dabei ist das letzte Feld (rechts unten) leer. Zu Beginn des Spieles werden die Buchstaben vermischt, das heißt, der Computer bewegt das freie Feld über das gesamte Spielfeld und vertauscht somit einen Buchstaben gegen den anderen. Wie oft ein Buchstabe mit einem anderen getauscht wird, bleibt dem Zufall überlassen.

Danach ist es Ihre Aufgabe, die Buchstaben wieder zu sortieren. Im Prinzip müssen Sie genauso arbeiten, wie der Computer vor Ihnen: Durch Vertauschen von Buchstaben mit dem freien Feld können Sie die Buchstaben verschieben. Und das müssen Sie solange tun, bis die Buchstaben von »A« bis »O« wieder sortiert sind.

Listing zu Buchstabendreher

```
10 REM ************************
20 REM ***                  ***
30 REM *** BUCHSTABENDREHER ***
40 REM ***                  ***
50 REM ************************
60 REM
70 GOSUB 1000
80 GOSUB 2500
90 GOSUB 3000
100 END
1000 REM
1010 REM *** VORBEREITUNGEN ***
1020 REM
1030 DEF FN X(X)=(X-1)*8+4
1040 DEF FN Y(X)=(X-1)*5+1
1050 DIM B(4,4)
1060 FOR I=1 TO 4
1070 :FOR J=1 TO 4
1080 : K=K+1
1090 : B(J,I)=K
1100 :NEXT J
1110 NEXT I
1120 DIM DI (4,2)
1130 FOR I=1 TO 4
1140 :READ DI (I,1),DI (I,2)
1150 NEXT I
1160 DATA 1,0,0,1,-1,0,0,-1
1170 PRINT "{CLR}";
1180 PRINT TAB(3);"{LIG.BLUE,RVSON,33SPACE}"
1190 FOR I=1 TO 4
1200 :FOR J=1 TO 4
1210 : PRINT "{3SPACE,RVSON,SPACE,RVOFF,7SPACE,RVSON,SPACE,RVOFF}";
1220 : PRINT "{7SPACE,RVSON,SPACE,RVOFF,7SPACE,RVSON,SPACE,RVOFF,7SPACE,
     RVSON,SPACE,RVOFF}"
1230 :NEXT J
1240 :PRINT "{3SPACE,RVSON,33SPACE}"
1250 NEXT I
1260 GOSUB 1500
1270 POKE 211,3:POKE 214,22:SYS 58640
1280 PRINT "<<ICH VERAENDERE DAS SPIELFELD>>"
1290 SX=4:SY=4
1300 SC=INT (RND(1)*50)+100
1310 FOR K=1 TO SC
1320 :D=INT (RND(1)*4)+1
1330 :PX=SX+DI(D,1):PY=SY+DI(D,2)
1340 :IF PX<1 OR PX>4 OR PY<1 OR PY>4 THEN GOTO 1320
1350 :B(SX,SY)=B(PX,PY)
1360 :J=SX:I=SY:GOSUB 1600
1370 :B(PX,PY)=16
1380 :SX=PX:SY=PY
1390 :J=SX:I=SY:GOSUB 1600
1400 NEXT K
1410 RETURN
1500 REM
1510 REM *** UNTERPROGRAMM 1 ***
1520 REM
1530 FOR I=1 TO 4
1540 :FOR J=1 TO 4
1550 : PRINT"{LIG.BLUE}";
1560 : GOSUB 1600
1570 :NEXT J
1580 NEXT I
1590 RETURN
```

```
1600 REM
1610 REM *** UNTERPROGRAMM 2 ***
1620 REM
1630 X=FN X(J)
1640 Y=FN Y(I)
1650 POKE 211,X:POKE 214,Y:SYS 58640
1660 PRINT "{LIG.BLUE}";
1670 IF B(J,I)<7 THEN ON B(J,I) GOSUB 1700,1750,1800,1850,1900,1950:RETURN
1680 ON B(J,I)-6 GOSUB 2000,2050,2100,2150,2200,2250,2300,2350,2400,2450
     :RETURN
1700 PRINT "{3SPACE}O{3SPACE,DOWN,6LEFT}";
1710 PRINT "{RVSON}Y{RVOFF}Y Y{RVSON}Y{RVOFF,SPACE,DOWN,6LEFT}";
1720 PRINT "{RVSON,SPACE}OOO{SPACE,RVOFF,SPACE,DOWN,6LEFT}";
1730 PRINT "Y{3SPACE}Y "
1740 RETURN
1750 PRINT " OOOO{2SPACE,DOWN,6LEFT}";
1760 PRINT "{RVSON,SPACE,RVOFF}OOO{RVSON}O{RVOFF,SPACE,DOWN,6LEFT}";
1770 PRINT "{RVSON,SPACE,RVOFF,3SPACE,RVSON,SPACE,RVOFF,SPACE,DOWN,6LEFT}";
1780 PRINT "{RVSON}OOOO{RVOFF,2SPACE}"
1790 RETURN
1800 PRINT " TTTTT{SPACE,DOWN,6LEFT}";
1810 PRINT "{RVSON,SPACE,RVOFF,5SPACE,DOWN,6LEFT}";
1820 PRINT "{RVSON,SPACE,RVOFF,5SPACE,DOWN,6LEFT}";
1830 PRINT "{RVSON}TTTTT{RVOFF,SPACE}"
1840 RETURN
1850 PRINT " TTTT{SPACE,DOWN,5LEFT}";
1860 PRINT "{RVSON,SPACE,RVOFF,3SPACE,RVSON,SPACE,RVOFF,SPACE,DOWN,6LEFT}";
1870 PRINT "{RVSON,SPACE,RVOFF,3SPACE,RVSON,SPACE,RVOFF,SPACE,DOWN,6LEFT}";
1880 PRINT "{RVSON}TTTT{RVOFF,2SPACE}"
1890 RETURN
1900 PRINT " TTTT{2SPACE,DOWN,6LEFT}";
1910 PRINT "{RVSON,SPACE,RVOFF}TT{2SPACE,DOWN,5LEFT}";
1920 PRINT "{RVSON,SPACE,RVOFF,4SPACE,DOWN,5LEFT}";
1930 PRINT "{RVSON}TTTT{RVOFF,2SPACE}"
1940 RETURN
1950 PRINT " TTTT{2SPACE,DOWN,6LEFT}";
1960 PRINT "{RVSON,SPACE,RVOFF}TT{2SPACE,DOWN,5LEFT}";
1970 PRINT "{RVSON,SPACE,RVOFF,4SPACE,DOWN,5LEFT}";
1980 PRINT "U{4SPACE}";
1990 RETURN
2000 PRINT " TTTT{2SPACE,DOWN,6LEFT}";
2010 PRINT "{RVSON,SPACE,RVOFF,SPACE}TT{SPACE,DOWN,5LEFT}";
2020 PRINT "{RVSON,SPACE,RVOFF,2SPACE,RVSON,SPACE,RVOFF,SPACE,DOWN,5LEFT}";
2030 PRINT "{RVSON}TTTT"
2040 RETURN
2050 PRINT " T{3SPACE}T{SPACE,DOWN,6LEFT}";
2060 PRINT "{RVSON,SPACE,RVOFF}TTT{RVSON,SPACE,RVOFF,SPACE,DOWN,6LEFT}";
2070 PRINT "{RVSON,SPACE,RVOFF,3SPACE,RVSON,SPACE,RVOFF,SPACE,DOWN,7LEFT}";
2080 PRINT " U{3SPACE}U ";
2090 RETURN
2100 PRINT "{2SPACE}OOO{2SPACE,DOWN,7LEFT}";
2110 PRINT "{3SPACE,RVSON,SPACE,RVOFF,2SPACE,DOWN,6LEFT}";
2120 PRINT "{3SPACE,RVSON,SPACE,RVOFF,2SPACE,DOWN,6LEFT}";
2130 PRINT "{2SPACE,RVSON}TTT{RVOFF,SPACE}"
2140 RETURN
2150 PRINT "{2SPACE}TTT{2SPACE,DOWN,6LEFT}";
2160 PRINT "{3SPACE,RVSON,SPACE,RVOFF,2SPACE,DOWN,6LEFT}";
2170 PRINT " T{SPACE,RVSON,SPACE,RVOFF,2SPACE,DOWN,5LEFT}";
2180 PRINT "{SPACE,RVSON}T{RVOFF,3SPACE}"
2190 RETURN
2200 PRINT " T{2SPACE}T{2SPACE,DOWN,6LEFT}";
2210 PRINT "{RVSON,SPACE,RVOFF}T{RVSON}T{RVOFF,3SPACE,DOWN,6LEFT}";
2220 PRINT "{RVSON,SPACE,RVOFF,SPACE,RVSON}T{RVOFF}T{2SPACE,DOWN,6LEFT}";
2230 PRINT "U{3SPACE}U "
2240 RETURN
```

```
225Ø PRINT " T{4SPACE,DOWN,5LEFT}";
226Ø PRINT "{RVSON,SPACE,RVOFF,5SPACE,DOWN,6LEFT}";
227Ø PRINT "{RVSON,SPACE,RVOFF,5SPACE,DOWN,6LEFT}";
228Ø PRINT "{RVSON}TTTT{RVOFF,2SPACE}"
229Ø RETURN
23ØØ PRINT " T{3SPACE}T{SPACE,DOWN,6LEFT}";
231Ø PRINT "{RVSON,SPACE}T{RVOFF}T{RVSON}T{SPACE,RVOFF,SPACE,DOWN,6LEFT}";
232Ø PRINT "{RVSON,SPACE,RVOFF,3SPACE,RVSON,SPACE,RVOFF,SPACE,DOWN,6LEFT}";
233Ø PRINT "U{3SPACE}U "
234Ø RETURN
235Ø PRINT " T{3SPACE}T{SPACE,DOWN,6LEFT}";
236Ø PRINT "{RVSON,SPACE}T{RVOFF}T{SPACE,RVSON,SPACE,RVOFF,SPACE,DOWN,6L
     EFT}";
237Ø PRINT "{RVSON,SPACE,RVOFF,2SPACE,RVSON}T{SPACE,RVOFF,SPACE,DOWN,6LEFT}";
238Ø PRINT "U{3SPACE}U "
239Ø RETURN
24ØØ PRINT " OOOOO{SPACE,DOWN,6LEFT}";
241Ø PRINT "{RVSON,SPACE,RVOFF,3SPACE,RVSON,SPACE,RVOFF,SPACE,DOWN,6LEFT}";
242Ø PRINT "{RVSON,SPACE,RVOFF,3SPACE,RVSON,SPACE,RVOFF,SPACE,DOWN,6LEFT}";
243Ø PRINT "{RVSON}TTTTT{RVOFF,SPACE}"
244Ø RETURN
245Ø PRINT "{7SPACE,DOWN,7LEFT}";
246Ø PRINT "{7SPACE,DOWN,7LEFT}";
247Ø PRINT "{7SPACE,DOWN,7LEFT}";
248Ø PRINT "{7SPACE}"
249Ø RETURN
25ØØ REM
251Ø REM *** SPIEL ***
252Ø REM
253Ø POKE 211,Ø:POKE 214,21:SYS 5864Ø
254Ø FOR I=1 TO 3
255Ø :PRINT "{4ØSPACE}";
256Ø NEXT I
257Ø POKE 211,Ø:POKE 214,21:SYS 5864Ø
2575 PRINT "{3SPACE}WELCHEN STEIN VERSCHIEBEN ?{3SPACE,2LEFT}";
258Ø GET A$:IF A$="" THEN GOTO 258Ø
259Ø PRINT A$
26ØØ IF A$>="A" AND A$<="O" THEN GOTO 264Ø
261Ø PRINT "{3SPACE}NUR BUCHSTABEN VON A BIS O !{3UP}"
262Ø FOR PA=1 TO 1ØØØ:NEXT PA
263Ø GOTO 2575
264Ø FOR K=1 TO 4
265Ø :PX=SX+DI(K,1):PY=SY-DI(K,2)
266Ø :IF PX<1 OR PX>4 OR PY<1 OR PY>4 THEN GOTO 268Ø
267Ø :IF B(PX,PY)=ASC (A$)-ASC("A")+1 THEN DG=DG+1:GOTO 272Ø
268Ø NEXT K
269Ø PRINT "{3SPACE}DIESER BUCHSTABE{2SPACE}GEHT NICHT !{3UP}"
27ØØ FOR PA=1 TO 1ØØØ:NEXT PA
271Ø GOTO 2575
272Ø B(SX,SY)=B(PX,PY)
273Ø J=SX:I=SY:GOSUB 16ØØ
274Ø B(PX,PY)=16
275Ø SX=PX:SY=PY
276Ø J=SX:I=SY:GOSUB 16ØØ
277Ø K=Ø
278Ø FOR I=1 TO 4
279Ø :FOR J=1 TO 4
28ØØ : K=K+1
281Ø : IF B(J,I)=K THEN NEXT J:NEXT I:RETURN
282Ø GOTO 253Ø
```

```
3000 REM
3010 REM *** ENDE ***
3020 REM
3030 PRINT "{CLR,2DOWN}SIE HABEN ES GELOEST !"
3040 PRINT "{2DOWN}DAS FELD WAR";SC;"MAL GEMISCHT."
3050 PRINT "{DOWN}FUER DIE LOESUNG HABEN SIE";DG
3060 PRINT "ZUEGE BENOETIGT."
3070 PRINT "{4DOWN}MOECHTEN SIE WEITERSPIELEN ?"
3080 GET A$:IF A$<>"J" AND A$<>"N" THEN GOTO 3080
3090 IF A$="J" THEN RUN
3100 RETURN
```

■ Programmbeschreibung für Buchstabendreher

Als erste Vorbereitung werden zwei Funktionen definiert, die – ähnlich wie bei »Fuenfer« – die Koordinaten für den Bildschirm berechnen. Diese Funktionen sind immer speziell auf das eine Programm zugeschnitten (und meistens durch Probieren »ausgefeilt« worden). Daher ist es nicht unbedingt erforderlich, daß Sie sie genau verstehen; wenn Sie die prinzipielle Arbeitsweise begriffen haben, können Sie die nächsten Programmzeilen in Angriff nehmen.

Zunächst wird ein Array definiert; dabei handelt es sich um ein zweidimensionales Array mit jeweils vier (um genau zu sein, fünf) Elementen. Sie erinnern sich doch, daß die Arrays auch die Elemente mit dem Index null benutzen können? Das wird zwar in den wenigsten Fällen ausgenutzt (auch hier nicht), aber ich möchte noch einmal darauf hingewiesen haben.

In der folgenden »Doppelschleife« wird dieses Feld auch gleich mit Werten belegt. Und zwar durchläuft Schleife (I) die Zeilen, während die innere Schleife (J) die einzelnen Spalten anspricht. Ganz innen wird eine Zählvariable (K) bei jedem Schleifendurchgang (alles in allem 16mal) um eins erhöht. Beim ersten Durchgang ist sie noch null. Diese Variable nimmt demnach im »Laufe der Schleifen« die Werte eins bis 16 an. Und genau diese Werte werden den einzelnen Elementen der Variable »B« zugeordnet.

Ich sagte vorhin, die Laufvariable »I« bediene die Zeilen, obwohl sie als äußerste Schleife (aus Gewohnheit) für die Spalten zuständig sein müßte. Sehen Sie sich dazu bitte Zeile 1090 an. Dort sind die beiden Variablen genau andersherum angeordnet, als man es »normalerweise« für diese Schleifenkonstruktion erwarten würde. Aber nur so läßt es sich bewerkstelligen, daß die Zahlen Zeile für Zeile zugeordnet werden. Sollte das noch etwas zu theoretisch sein, betrachten Sie bitte den Bildschirm, direkt nach dem Starten des Programms. Da sehen Sie, daß die Buchstaben quer angeordnet sind, das heißt, daß das »B« neben und nicht unter dem »A« steht. Und in unserem Programm werden die Buchstaben lediglich durch Zahlen vertreten.

Mit den nächsten drei (bzw. vier) Zeilen wird eine Variable namens »DI« angelegt; sie enthält Werte für das Verändern von Koordinaten. Das klingt im Moment sicher abschreckend theoretisch, aber Sie werden an einer anderen Stelle noch sehen, wie einfach man mit dieser Variablen arbeiten kann.

Den READ-Befehl kennen Sie bereits. Und Sie wissen auch, daß dahinter eine Variable stehen muß, der ein Wert aus der DATA-Liste zugewiesen wird. Hier dagegen sehen Sie zum ersten Mal, daß es nicht nur ein Wert sein muß, der dem READ-Befehl folgt; es können mehrere Variablen sein.

Beispiel:

```
DATA 1, 2, 3, 4, 5, 6
READ A, B, C, D, E, F
```

Einfacher geht es mit einer Feldvariablen (zumindest bei größeren Datenmengen).

```
DATA 1, 2, 3, 4, 5, 6
FOR I=1 TO 6
:READ A(I)
NEXT I
```

Einen Fehler sollten Sie allerdings nicht begehen: Sie dürfen nicht vergessen, die Variable »A« zu indizieren. Angenommen, Ihr Programm sähe so aus:

```
DATA 1, 2, 3, 4, 5, 6
FOR I=1 TO 6
:READ A
NEXT I
```

Falls Sie die Werte von »A« ausgeben lassen wollen, wundern Sie sich nicht, daß »A« nur den Wert 6 aufweist. Für den Computer stellt die obige Anweisung nur mehrere Wertzuweisungen an ein und dieselbe Variable dar. Daher kann er sie nicht unterscheiden, das bedeutet, er überschreibt bei jedem Schleifendurchgang den alten Wert mit dem neuen. Und der neueste Wert ist nach der Schleife eben sechs.

An noch etwas möchte ich Sie an dieser Stelle erinnern: Die DATA-Zeilen müssen nicht unbedingt vor dem READ-Befehl stehen. Sie können zum Beispiel ganz am Schluß an ein Programm angehängt werden, wo der Interpreter nie hinspringt (mittels GOTO oder ähnlichem). Dennoch wird der Computer die DATA-Zeilen mit seinem Adreßzeiger finden, der ja auf das erste DATA-Element zeigt.

In diesem Minibeispiel konnte darauf verzichtet werden, A(I) zu dimensionieren. Zum einen, weil dieses Beispiel kein richtiges Programm ist, und zum anderen, da sich die Indizes für »A« zwischen eins und zehn bewegen. Und solange das der Fall ist, braucht man »A« nicht zu dimensionieren. Man sollte es zwar des Programmierstils wegen tun, es muß aber – wie gesagt – nicht sein.

Die Zeilen 1170 bis 1250 bauen den Rahmen für das Spielfeld auf. Dafür wird zuerst der Bildschirm gelöscht. Da die Ausgabe noch in der obersten Zeile erfolgen soll, muß hinter dem [SHIFT]+[CLR/HOME] ein Strichpunkt stehen.

In Zeile 1180 wird ein Strich quer über den Bildschirm gezogen, Dazu wird beginnend bei Spalte drei einfach eine Zeile invers dargestellter [SPACE] ausgegeben. In der geschachtelten Schleife werden die Trennwände zwischen den einzelnen Buchstabenfeldern gezogen (1210 und 1220). Nach jeder vollendeten Zeile (sprich, wenn die J-Schleife durchlaufen worden ist), zieht die Zeile 1240 nocheinmal eine Linie über den Bildschirm, um die

nächste Zeile Buchstaben abzutrennen. Daraufhin kommt das NEXT für die äußerste Schleife.

Das aufgerufene Unterprogramm ab Zeile 1500 wird später noch besprochen. Daher möchte ich an dieser Stelle nur verraten, daß dieses Unterprogramm für die Ausgabe der Buchstaben zuständig ist. Das bedeutet, daß der Bildschirm vollständig aufgebaut ist, sobald der Computer die Zeile 1270 erreicht.

Dabei wird hier nur der Cursor an eine andere Stelle gesetzt und ein erklärender Text ausgegeben.

In den Zeilen 1290 bis 1400 werden die Buchstaben schön vermischt. Dazu muß der Computer erst einmal wissen, wo sich das freie Feld gerade befindet, mit dessen Hilfe die Buchstaben nur sortiert werden können (hier ergeben sich ein paar Parallelen mit dem Programm »Faßfarben«). Und dieses leere Feld befindet sich zu Beginn ganz rechts unten, das heißt, seine Koordinaten sind 4/4 (SX und SY).

Mit »SC« wie (SChleife) wird festgelegt, wie oft Buchstaben miteinander vertauscht werden sollen, wie gut das Brett also gemischt sein soll. SC kann Werte zwischen 100 und 149 annehmen.

In Zeile 1310 sehen Sie auch gleich, daß SC wirklich als obere Grenze für die Schleife dient. Mit Hilfe der nächsten zwei Zeilen wird die Bewegung des freien Feldes (zufalls)gesteuert. Jetzt kommen wir auch wieder auf die Variable DI vom Anfang zu sprechen. Doch zuerst wird in 1320 der Variablen »D« ein zufälliger Wert zugewiesen. Er kann Werte zwischen eins und vier annehmen und hat einige Bedeutung für »DI« (daher D).

Wenn Sie sich die DATA-Zeile in 1160 noch einmal ansehen wollen: Man kann diese acht Werte in vier Wertepaare unterteilen, das ergibt also: (1/0) (0/1) (−1/0) (0/−1). Dabei stellte die erste Zahl die Veränderung der X-Koordinate, die zweite die der Y-Koordinate dar. Wenn man diese Wertepaare zu »X/Y« addiert, wird einer der beiden Werte um eins vergrößert bzw. verkleinert. Was mit welchem Wert geschieht, entscheidet die Zufallsvariable »D«. Betrachten Sie dazu bitte Zeile 1330.

Zu »SX« und »SY« wird jeweils DI(D,1) und DI(D,2) addiert. Und das entspricht genau der Veränderung, die ich eben angesprochen habe. Aufgrund der Werte aus den DATA-Zeilen ist ersichtlich, daß sich immer nur der X- oder der Y-Wert um eins verändern kann. Ausgeschlossen ist, daß sich das freie Feld schräg bewegt (wenn sich beide Koordinaten ändern), und daß es sich gleich um mehrere Spalten und Zeilen verschiebt. Allerdings werden diese neuen Werte nicht sofort den Variablen SX/SY zugewiesen. Zunächst muß der neue Wert auf seine Richtigkeit überprüft werden (Zeile 1340).

Ist alles in Ordnung, wird dem ehemals freien Feld B(SX, SY) der Wert des danebenliegenden Feldes B(PX, PY) zugewiesen, das ab sofort frei sein soll. In Zeile 1360 (auch in 1390) wird lediglich eine Ausgaberoutine aufgerufen, die die einzelnen Veränderungen jeweils ausgeben soll. Doch dazu später mehr.

Dem neuen leeren Feld wird der Wert 16 zugewiesen, damit es als solches auch vom Programm erkannt wird (da die ersten 15 Buchstaben des Alphabets verwendet werden, kann man mit 16 das einzige freie Feld definieren). Würde man ihm keinen neuen Wert zuweisen, wäre ein Buchstabe doppelt, da der Inhalt dieser Variable in Zeile 1350 in B(SX, SY) kopiert wurde. Zum Abschluß dieser Schleife müssen nur noch die »computerinternen« Koordinaten des freien Feldes aktualisiert werden (Zeile 1390), bevor der NEXT-Befehl die Schleife abschließt. Wurde die Schleife vollständig durchlaufen, kann das Programm mit dem RETURN-Befehl wieder zurückkehren.

Ab Zeile 1500 steht ein Unterprogramm, das seinerseits wieder ein Unterprogramm benutzt. Ab 1600 liegt eine Routine, die einen einzelnen Buchstaben auf dem Bildschirm ausgibt. Dazu müssen dieser Routine die Koordinaten in Form von »J« (für X) und »I« (für Y) übergeben werden. Dann berechnet sie sich selbständig die dazugehörige Bildschirmposition und gibt den entsprechenden Buchstaben aus.

Damit alle Buchstaben ausgegeben werden, benötigt man nicht mehrere komplette Ausgaberoutinen, sondern benutzt zwei Schleifen, möglichst mit »I« und »J« als Laufvariablen, sowie die eigentliche Ausgaberoutine (in diesem Fall ab 1600). Sehen Sie sich das in der Praxis an: In den Zeilen 1530 bis 1580 steht eine verschachtelte Schleife, die für jeden Durchlauf den dazugehörigen Buchstaben zeichnen läßt.

Somit hätten wir schon das ganze Geheimnis um Unterprogramm eins gelüftet; jetzt geht es nur noch um die zweite Routine: Zunächst berechnet sie die wirklichen Bildschirmpositionen (1630 und 1640). Anschließend wird gleich einmal der Cursor an diese Stelle gesetzt (Zeile 1650). Zeile 1660 setzt lediglich die Farbe fest, mit der geschrieben werden soll.

In 1670 dagegen sehen Sie eine monströse Zeile. Sie beginnt mit einer Abfrage, und zwar, ob der Inhalt der Variablen »B« kleiner sieben sei. Damit wird geprüft, ob der darzustellende Buchstabe unter den ersten sechs liegt (beim siebten ist die Bedingung bereits nicht mehr erfüllt!). Ist das der Fall, dann kann das Programm mit einem »ON ... GOSUB« alle einzelnen PRINT-Routinen anspringen, die letztendlich für die Ausgabe eines einzelnen Buchstabens verantwortlich sind. Handelte es sich bei B(J,I) um einen der anderen neun Buchstaben bzw. um das leere Feld, muß der Computer die »ON ... GOSUB«-Liste in Zeile 1680 abklappern. Davor muß er jedoch noch von »B« sechs subtrahieren, damit der Computer auch die richtige Zeile anspringt.

Ich möchte Ihnen das an einem konkreten Beispiel näher erläutern. Nehmen wir an, »J« wäre drei und »I« gleich zwei. Mit dieser Voraussetzung kommt der Computer bei Zeile 1630 an. Dort wird zunächst die Funktion »X« aufgerufen, die der Variablen »X« den Wert 20 zuweist (einfach »J« in 1030 eingesetzt und ausgerechnet). Für »Y« ergibt sich ein Wert von sechs. Der Cursor wird also an die 20ste Spalte in der sechsten Zeile gesetzt.

Im folgenden nehmen wir an, die Variable »B« hätte für J=3 und I=2 den Buchstaben »D« auf Lager. »D« würde durch die Zahl 4 repräsentiert. Demnach wäre die Bedingung in 1670 erfüllt. Jetzt durchsucht der Interpreter die Zeilenliste, bis er auf den vierten Wert

stößt, das ist 1850. Also springt er mit einem GOSUB zu Zeile 1850, gibt dort die Zeichen aus, trifft auf RETURN und kehrt zurück. Dabei kommt er wieder in Zeile 1670 und findet hinter den ganzen Zeilennummern doch noch einen Befehl, RETURN; also kehrt er jetzt ganz zurück, wo er herkam.

Nehmen wir jetzt an, »B« hätte für dieselben »I« und »J« den Wert neun (für den Buchstaben »I«). Somit wäre die Bedingung in 1670 nicht mehr erfüllt. Jetzt springt der Computer in Zeile 1680. Dort subtrahiert er von neun sechs, das ist drei, und findet als dritten Wert in der Reihe 2100, springt dorthin...

Nehmen wir an, dieses »minus sechs« in 1680 würde fehlen. Dann suchte der Computer in der Liste nach dem neunten Wert, das ergäbe 2400 für »O«, und das wäre sicherlich nicht erwünscht, oder? Die einzelnen Ausgaberoutinen sind nichts Weltbewegendes. Die Buchstaben sind durch Grafikzeichen zusammengesetzt, damit sie größer dargestellt werden können. Das einzig Erwähnenswerte ist vielleicht, warum so viele Cursor-Bewegungen mit [CRSR-DOWN] und [CRSR-LEFT] stattfinden. Man hätte natürlich vor jede einzelne PRINT-Anweisung noch einmal die Folge von POKE- und SYS-Befehl setzen können, damit der Cursor immer am richtigen Fleck ist. Zum einen wäre das aber in eine »Wahnsinns-Tipparbeit« ausgeufert, und zum anderen ist der POKE-Befehl einer der langsamsten im C64-Basic überhaupt. Darum wurden die vielen Steuerzeichen in Kauf genommen.

Aber etwas anderes möchte ich Ihnen noch erklären. Sehen Sie sich zunächst einmal an, wie verschachtelt die GOSUB und RETURN-Befehle unter Umständen arbeiten:

Zeile 70 springt nach 1000
: 1260 ruft 1500 auf
:   von 1500 wird nach 1600 verzweigt
:   dort wird noch einmal die eigentliche Ausgabe aufgerufen

Sie sehen, vier GOSUBs können verschachtelt sein. Jedes einzelne RETURN muß natürlich erst einmal die Zeilennummer suchen, zu der es zurück muß. Und jeder GOSUB-Befehl sucht die Zeile, zu der er springen muß *und* speichert auch noch die Zeilennummer, von der er losgeschickt wurde. Man könnte das Unterprogramm ab 1600 ein klein bißchen vereinfachen, indem Sie ganz einfach die GOSUB-Befehle in 1670 und 1680 durch GOTOs ersetzen und die RETURNs an den Zeilenenden ersatzlos streichen. Was passiert dann?

Sobald Zeile 1600 angesprungen wird, wird die Ausgangszeile gespeichert. Jetzt wird in Zeile 1670 und 1680 mittels GOTO zu den kleinen Ausgaberoutinen gesprungen, die immer noch mit einem RETURN jeweils abgeschlossen werden; das bedeutet also nichts anderes, als daß der Computer sich seine letzte gespeicherte Zeilennummer aus dem Stack holt und dahin zurückkehrt; und das ist nichts anderes als die eben erwähnte Ausgangszeile. Das bringt Ihnen höhere Geschwindigkeit, weniger Befehle ...

Aber das können Sie sich wirklich für die Zukunft merken: Sobald auf einen GOSUB-Befehl unmittelbar ein RETURN-Befehl folgt (siehe Zeile 1670), kann das GOSUB durch ein GOTO ersetzt und das RETURN ersatzlos gestrichen werden!

Die Ausgabe der Buchstaben wurde großteils durch Ausprobieren erreicht. Ich glaube, es ist keine Schande, das zuzugeben. Schließlich kann man die Ausgabe nur am Bildschirm optimieren. Daher dürfte es auch nicht allzuviel bringen, wenn ich Ihnen alle Grafikzeichen im einzelnen erläutere.

Aber schauen Sie sich bitte einmal die Vorderseite der Tastatur auf Ihrem C64 an. Dort sehen Sie verschiedene Grafikzeichen. Wie Sie bereits wissen (sollten), erreichen Sie das jeweils rechte mit [SHIFT], das linke mit [CTRL] und der entsprechenden Taste. Drücken Sie also beispielsweise [SHIFT]+[S], dann erhalten Sie ein Herz. Wenn Sie aber [CTRL]+[K] drücken, erscheint ein vertikaler »Balken« an der Cursor-Position.

Mit diesen Grafikzeichen wurde die Ausgabe programmiert. Sollten Sie künstlerische Ambitionen haben, bleibt es Ihnen natürlich ungenommen, die Ausgabe noch zu verfeinern! Dann kommen wir jetzt zum Programmteil »SPIEL«. Die Zeilen 2530 bis 2575 sind mal wieder nur für die Ausgabe zuständig. Und zwar wird zunächst der Cursor in die 21. Bildschirmzeile gesetzt. Die FOR-NEXT-Schleife löscht drei Bildschirmzeilen, und zwar die Zeilen 21, 22 und 23. Daraufhin wird der Cursor noch einmal an die Zeile 21 gesetzt, und in dieser Zeile erfolgt die Frage, welchen Stein Sie verschieben möchten. Mit Stein ist natürlich ein Buchstabe gemeint, aber da wir es hier mit Spielsteinen zu tun haben ...

Die Zeile 2580 sollte klar sein. Wie üblich, wird eine Leereingabe sofort abgefangen. Sollten Sie jedoch eine Taste gedrückt haben (egal welche), wird diese mit »PRINT A$« in Zeile 2590 ausgegeben. Erst in 2600 erfolgt die Überprüfung, ob Ihre Eingabe im erlaubten Bereich war. Ist alles in Ordnung, überspringt der Computer die nächsten drei Zeilen. Andernfalls werden gerade diese drei Zeilen bearbeitet, die eine Fehlermeldung ausgeben, eine kurze PAuse (Variable PA) einlegen und anschließend zu der Eingabe in 2575 zurückkehren.

Mit der nächsten Schleife (von Zeile 2640 bis 2680) wird überprüft, ob Ihr eingegebener Buchstabe auch wirklich neben dem freien Feld liegt. Nur in diesem Fall können Sie den Buchstaben wirklich verschieben!

Dazu benutzt das Programm wieder die Variable DI. Und zwar wird zu SX und SY (der Position des freien Feldes) jeweils DI(K,0) und DI(K,1) addiert. Da »K« die Werte eins bis vier annehmen kann, werden im Laufe der Schleife alle Möglichkeiten ausprobiert.

Sollte nach der Addition die Variablen PX oder PY einen unmöglichen Wert haben (nämlich außerhalb des Spielfeldrandes liegen), wird sofort zum NEXT-Befehl verzweigt, um die nächste Richtung auszuprobieren (Zeile 2660). Sind PX und PY zulässig, wird noch geprüft, ob dieser Nachbarbuchstabe mit Ihrer Eingabe übereinstimmt. Falls ja, können ab 2720 die Felder vertauscht werden. Andernfalls kommt das Programm wieder zur NEXT-

Schleife. Sollte das Programm bei allen K-Werten auf das NEXT gestoßen sein, bedeutet das, daß der Buchstabe, den Sie eingegeben haben, nicht neben dem leeren Feld liegt.

Es dürfte das Beste sein, das Ganze an einem konkreten Fall noch einmal nachzuvollziehen. Gehen wir dabei von folgendem Fall aus:

```
  1  2  3  4
1 ...B..F..D
2 ...E.. ..G
3 ...H..J..K
4 ..........
```

Die Punkte sollen das restliche Spielfeld andeuten. Zum besseren Verständnis habe ich auch die Zeilen- und Spaltennummern eingetragen. Bei 3/2 sehen Sie das leere Feld (zwischen »E« und »G« bzw. »F« und »J«).

Nehmen wir an, Sie hätten als Buchstaben »J« eingegeben. Dafür müßte die X-Koordinate gar nicht verändert, die Y-Koordinate des leeren Feldes um eins vergrößert werden. Das entspräche der Variablen DI (2,1) und DI (2,2) (siehe Zeile 1160).

In Zeile 2640 beginnt jetzt die Schleife, »K« ist eins. Dadurch wird die X-Koordinate um eins erhöht, »Y« bleibt gleich, man erhält als Inhalt von B(PX,PY) den Buchstaben G. PX ist 4, PY ist 2. Demnach sind die Bedingungen in 2660 nicht erfüllt, das Programm kommt nach 2670. Um die Funktionsweise dieses Programms besser verstehen zu können, muß ich etwas weiter ausholen:

Sie haben bereits gelernt, daß Sie ein Zeichen auf den Bildschirm »POKEn« können, zum Beispiel POKE 1024,1. Die eins repräsentiert das A. Intern werden die Zeichen als Zahlen gespeichert. Jetzt gibt es aber noch eine zweite Zahl für jeden Buchstaben, den sogenannten ASCII-Code (»**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange«). Dabei wurde international festgelegt, welche Zahl welchen Buchstaben darstellen soll, und jeder Computerhersteller hält sich mehr oder weniger an diesen Code. Dabei wurde nun einmal festgelegt, daß das »A« nicht durch eine eins, sondern durch eine 65 dargestellt werden soll. Geben Sie beispielsweise ein:

```
PRINT CHR$(65)
```

Als Ergebnis werden Sie »A« erhalten. Möchten Sie dagegen diese Zahl für einen bestimmten Buchstaben ausgegeben haben, geben Sie folgendes ein:

```
PRINT ASC("A")
```

Diesmal wird auf dem Bildschirm 65 erscheinen. Und genau diese Funktion wird in Zeile 2670 benutzt. Das Programm berechnet von Ihrer Eingabe den ASCII-Code (ASC(A$)). Da unser Programm aber bei eins zu zählen beginnt und nicht bei 65, muß zur Überprüfung der Buchstaben 64 subtrahiert werden. Das kann man ganz elegant lösen, indem man den ASCII-Code für »A« subtrahiert und eins addiert.

Beispiel:

Nehmen wir an, Sie geben ein [A] ein. Dann ist ASC("A") gleich 65, minus ASC("A") ergibt null, plus eins führt zu dem Ergebnis eins. Die eins stellt in unserem ganz speziellen Programm das »A« dar. Geben Sie dagegen ein [B] ein, kommt die Berechnung in 2670 zu dem Ergebnis zwei. Sie sehen, dieser »Algorithmus« funktioniert einwandfrei. Stimmt also Ihre Eingabe mit dem Inhalt des einen Nachbarfeldes überein, ist die Bedingung in 2670 erfüllt, und das Programm kann nach 2720 verzweigen.

In unserem Beispiel haben Sie »J« eingegeben und das Feld B(PX,PY) enthält den Wert für »G«. Also kann die Bedingung nicht erfüllt sein, das Programm trifft demnach auf den NEXT-Befehl.

Diesmal wird in 2650 nicht die X-, sondern die Y-Koordinate um eins erhöht, das ergibt den Buchstaben [J]. Jetzt können Sie die beiden nächsten Zeilen einmal selbst durchgehen. Sicher ist auf jeden Fall, daß die Bedingung in 2670 erfüllt ist; daher wird DG (wie DurchGang) um eins erhöht und dann zu Zeile 2720 verzweigt.

Hätten Sie bei der Frage in 2575 ein [E] eingegeben, wäre das erst bei K=3, ein [F] gar erst bei K=4 gefunden worden. Vertauschen Sie aber bitte in Gedanken einmal das leere Feld mit dem Buchstaben G. In dem Fall wird bei K=1 zu SX, das bereits vier ist, noch eins addiert (SX+DI(K,1)). In diesem Fall läge die neue Koordinate PX mit fünf eindeutig außerhalb des Feldes. Um einen solchen Unsinn zu vermeiden, wurde die Abfrage in 2660 eingefügt.

Hätten Sie bei Ihrer Eingabe ein [A] eingegeben, wäre die K-Schleife viermal umsonst durchlaufen worden und daraufhin die Zeilen 2690 bis 2710 bearbeitet worden. Nehmen wir aber noch einmal den Fall her, daß Sie ein [J] eingegeben haben und dies bei K=3 erkannt worden wäre. Dann wird der Variable B(3,2) der Inhalt des Feldes B(3,3) zugeordnet (Zeile 2720). In diesem Moment ist »J« doppelt vorhanden, sowohl in B(3,2) als auch in B(3,3). Doch zunächst wird die Ausgaberoutine angesprungen. In diesem Fall wird das »J« über das leere Feld geschrieben. Danach wird dem ehemaligen J-Feld B(3,3) der Wert für das leere Feld (16) zugewiesen; anschließend müssen noch die Koordinaten für das leere Feld neu gesetzt werden, bevor die Ausgaberoutine aufgerufen werden kann; diesmal wird das »J« durch das freie Feld überschrieben.

Mit der anschließenden K-Schleife wird überprüft, ob Sie das Feld bereits vollständig sortiert haben. Bei jedem Durchlauf der Schleifen wird überprüft, ob der Inhalt des Feldes »B« mit dem Sollwert übereinstimmt. Schon bei der ersten Abweichung bearbeitet der Interpreter die Zeile 2820, das heißt, die beiden Schleifen von 2780 und 2790 werden niemals mehr vollendet werden ...

Sollte aber bei allen Überprüfungen festgestellt worden sein, daß das Feld bereits in der richtigen Reihenfolge vorliegt, trifft das Programm auf den RETURN-Befehl, sprich, das Programm neigt sich dem Ende zu (wie auch diese Beschreibung). Denn den Rest kann ich mir sparen!

10 Würmli

Einführung

Dieser Name steht für ein ganz niedliches Programm, Spiel stimmt in diesem Fall eigentlich gar nicht. Es geht nämlich darum, ein Labyrinth zu konstruieren, durch das Würmli versucht, selbständig wieder herauszukommen. Im »Eingabeteil« wird Ihnen zwar am unteren Bildschirmrand immer eingeblendet, welche Tasten Sie benutzen können; dennoch möchte ich Sie mit dem Editor vertraut machen.

Am Rande: Ein Editor ist ein Programm, mit dem man entweder Texte oder Programme oder ähnliches editiert, das heißt bearbeitet. Zum Beispiel könnte man den Teil Ihres C64, in dem Sie Ihre Basic-Programme eingeben, als Editor bezeichnen. In unserem Fall bearbeiten Sie ein Labyrinth.

Auf dem Bildschirm sehen Sie eine Umrandung. Sie begrenzt das Labyrinth. In diesem Rahmen können Sie (fast) unbegrenzt walten. Dabei stellt Ihnen das Programm einige Hilfsfunktionen zur Verfügung.

Wenn Sie den Cursor nur über den Bildschirm »laufen« lassen wollen, drücken Sie [P]. Anschließend können Sie Ihre Schreibmarke mit den Tasten [R], [L], [H] und [T] für **r**echts, **l**inks, **h**och und **t**ief hin- und herbewegen.

Möchten Sie Mauerteile einzeichnen, drücken Sie [Z]. Hierauf können Sie wieder dieselben Tasten wie oben verwenden. Allerdings hinterläßt der Cursor diesmal eine »Spur«, das heißt, an jeder Stelle wird ein Mauerstück gesetzt. Haben Sie sich völlig zugebaut und wollen wieder bestimmte Teile löschen, dann drücken Sie [W]. Auch hier gilt dasselbe wie oben (hinsichtlich der Tasten).

Während Sie das Labyrinth entwerfen, können Sie natürlich ständig zwischen diesen verschiedenen Modi wechseln. Sie müssen nicht erst fertigzeichnen, um am Schluß alle Versehen wieder zu löschen. Sondern sollten Sie einen Fehler gefunden haben, können Sie mit [P] das Zeichnen beenden, an die betreffende Stelle fahren und mit [W] die überflüssigen Mauerteile löschen.

Sind Sie aber zu der Einsicht gekommen, es wäre besser, gleich noch einmal von vorne anzufangen, drücken Sie [N]. Daraufhin wird das Feld gelöscht, und Sie können wieder beginnen.

Sollten Sie dagegen das Programm abbrechen wollen, brauchen Sie bloß [A] zu betätigen. Sofort erscheint eine Sicherheitsabfrage, ob Sie das Programm wirklich verlassen möchten. Falls Sie mit [N] antworten, kommen Sie wieder zurück in den Editor, andernfalls bricht der C64 ab.

Der letzte Menüpunkt startet Würmli, mit [G] für »GO«. In der linken oberen Ecke erscheint Ihr Gefährte (in Form eines kleinen Herzens; ist er nicht wirklich lieb?). Er sucht sich jetzt seinen Weg durch das Labyrinth. Es dürfte nicht oft gelingen, den guten Würmli in die Irre laufen zu lassen, bis er gar nicht mehr weiß, wo er ist. Aber vielleicht ist gerade das ein Ansporn für Sie, ein möglichst schweres Labyrinth zu konstruieren?

Listing zu Würmli

```
1Ø REM ***************
2Ø REM ***         ***
3Ø REM *** WUERMLI ***
4Ø REM ***         ***
5Ø REM ***************
6Ø REM
7Ø GOSUB 1ØØØ
8Ø GOSUB 2ØØØ
9Ø GOSUB 3ØØØ
1ØØ END
1ØØØ REM
1Ø1Ø REM*** VORBEREITUNGEN ***
1Ø2Ø REM
1Ø25 POKE 5328Ø,Ø:POKE 53281,Ø
1Ø3Ø PRINT "{CLR,BLUE,RVSON,4ØSPACE,RVOFF}";
1Ø35 PRINT  "{BLACK,4ØSPACE}";
1Ø4Ø FOR I=1 TO 19
1Ø5Ø :PRINT "{BLUE,RVSON,SPACE,RVOFF,BLACK,38SPACE,BLUE,RVSON,SPACE}";
1Ø6Ø NEXT I
1Ø7Ø PRINT"{RVSON,4ØSPACE}";
1Ø8Ø PRINT "{WHITE,RVSON}P)OSITION R)ECHTS{3SPACE}L)LINKS H)HOCH T)IEF ";
1Ø9Ø PRINT "{RVSON}Z)EICHNEN W)LOESCHEN N)EU A)USGANG G)Ø{SPACE,HOME}";
11ØØ POKE 65Ø,128
111Ø FA$="{BLACK,WHITE,RED,CYAN,PURPLE,GREEN,BLUE,YELLOW,ORANGE,BROWN,LI
     G.RED,GREY 1,GREY 2,LIG.GREEN,LIG.BLUE,GREY 3}"
112Ø IX=1:IY=1:X2=1:Y2=1
113Ø POKE 211,IX:POKE 214,IY:SYS 5864Ø
114Ø PRINT "{RVSON,CYAN,SPACE}";
115Ø FR$="":FB$="{RVSON,BLACK}"
116Ø GET AN$:IF AN$="" THEN GOTO 116Ø
117Ø IF AN$="G" THEN GOTO 13ØØ
118Ø IF AN$="Z" THEN GOTO 135Ø
119Ø IF AN$="P" THEN GOTO 14ØØ
12ØØ IF AN$="W" THEN GOTO 145Ø
121Ø IF AN$="N" THEN GOTO 15ØØ
122Ø IF AN$="A" THEN GOTO 155Ø
123Ø GOTO 111Ø
13ØØ REM **************************** GO
131Ø RETURN
135Ø REM ********************** ZEICHNEN
136Ø FR$="{BLUE}":GOTO 16ØØ
14ØØ REM ********************** POSITION
141Ø FR$="":GOTO 16ØØ
145Ø REM ********************** LOESCHEN
146Ø FR$="{BLACK}":GOTO 16ØØ
15ØØ REM ************************** NEU
151Ø GOTO 1ØØØ
155Ø REM *********************** AUSGANG
156Ø GOSUB 4ØØØ
157Ø PRINT "MOECHTEN SIE DAS SPIEL BEENDEN (J/N) ?"
158Ø GET AN$:IF AN$="" THEN GOTO 158Ø
159Ø IF AN$="J" THEN END
1595 GOSUB 4ØØØ:GOTO 1Ø8Ø
16ØØ REM ******************* RICHTUNGEN
161Ø GET AN$:IF AN$="" THEN GOTO 161Ø
162Ø IF AN$="R" THEN GOTO 17ØØ
163Ø IF AN$="T" THEN GOTO 175Ø
164Ø IF AN$="L" THEN GOTO 18ØØ
165Ø IF AN$="H" THEN GOTO 185Ø
166Ø GOTO 117Ø
```

```
1700 REM ************************ RECHTS
1710 X2=IX+1:IF X2<39 THEN GOTO 1900
1720 X2=38:GOTO 1900
1750 REM ************************** TIEF
1760 Y2=IY+1:IF Y2<21 THEN GOTO 1900
1770 Y2=20:GOTO 1900
1800 REM ************************ LINKS
1810 X2=IX-1:IF X2>0 THEN GOTO 1900
1820 X2=1:GOTO 1900
1850 REM ************************** HOCH
1860 Y2=IY-1:IF Y2>0 THEN GOTO 1900
1870 Y2=1
1900 PRINT FR$;:IF FR$="" THEN PRINT FB$
1910 POKE 211,IX:POKE 214,IY:SYS 58640
1920 PRINT "<RVSON,SPACE>";
1930 FB$=MID$(FA$,1+(PEEK(55296+X2+40*Y2) AND 15), 1)
1940 POKE 211,X2:POKE 214,Y2:SYS 58640
1950 PRINT "<CYAN,RVSON,SPACE>";
1960 IX=X2:IY=Y2:GOTO 1600
2000 REM
2010 REM *** SPIEL ***
2020 REM
2030 GOSUB 4000
2040 PRINT " <<< WUERMLI SUCHT SICH SEINEN WEG >>>";
2050 X2=1:Y2=1
2060 POKE 211,X2:POKE 214,Y2:SYS 58640
2070 PRINT "<GREEN>S̲";
2080 PRINT FR$;:IF FR$="" THEN PRINT FB$
2090 POKE 211,IX:POKE 214,IY:SYS 58640
2100 PRINT "<RVSON,SPACE>";
2110 X3=1:Y3=0:RI=4
2120 XX=X2+X3:YY=Y2+Y3
2130 IF XX=0 AND YY=1 THEN RETURN
2140 IF XX=39 AND YY=1 THEN RETURN
2150 IF XX<1 OR XX>38 OR YY<1 OR YY>20 THEN GOTO 2180
2160 FA=PEEK(55296+XX+40*YY) AND 15
2170 IF FA=0 THEN GOTO 2230
2180 RI=RI-1:IF RI<1 THEN RI=4
2190 IF RI=4 THEN X3=1:Y3=0:GOTO 2120
2200 IF RI=3 THEN X3=0:Y3=1:GOTO 2120
2210 IF RI=2 THEN X3=-1:Y3=0:GOTO 2120
2220 IF RI=1 THEN X3=0:Y3=-1:GOTO 2120
2230 POKE 211,X2:POKE 214,Y2:SYS 58640
2240 PRINT "<RVOFF,WHITE,SPACE>";
2250 X2=X2+X3:Y2=Y2+Y3
2260 POKE 211,X2:POKE 214,Y2:SYS 58640
2270 PRINT "<RVOFF,GREEN>S̲";
2280 CL=CL+1:RI=RI+1:IF RI>4 THEN RI=1
2290 GOTO 2190
3000 REM
3010 REM *** ENDE ***
3020 REM
3030 POKE 211,X2:POKE 214,Y2:SYS 58640
3040 PRINT "<RVSON,BLACK,SPACE>";
3050 X2=X2+X3:Y2=Y2+Y3
3060 FOR K=1 TO 10
3070 :POKE 211,X2:POKE 214,Y2:SYS 58640
3080 :PRINT "<GREEN,RVOFF>S̲";
3090 :PRINT "<LEFT,BLACK,SPACE>";
3100 NEXT K
3120 GOSUB 4000
3130 IF X2<>39 THEN GOTO 3170
3140 PRINT "<DOWN,WHITE,SPACE><<< WUERMLI HAT DEN WEG GEFUNDEN >>>"
3150 PRINT "ER FAND DEN AUSGANG IN";CL;"CLICK ..."
3160 RETURN
```

```
3170 PRINT "WUERMLI FINDET DEN AUSGANG NICHT."
3180 PRINT "ER STEHT WIEDER AM ANFANG ..."
3190 RETURN
4000 REM
4010 REM ** LOESCHEN DER UNTEREN ZEILEN
4020 REM
4030 POKE 211,0:POKE 214,22:SYS 58640
4040 PRINT "{40SPACE}";
4050 PRINT "{40SPACE}";
4060 PRINT "{2UP}";
4070 RETURN
```

Programmbeschreibung für Würmli

Wie so viele andere Programme vorher, beginnt auch dieses Spiel mit dem Bildschirmaufbau. Zunächst werden die Bildschirmfarben eingestellt. Dazu muß man wissen, daß die Speicherstellen 53280 und 53281 die Farben für den Rahmen und den Hintergrund enthalten. Wenn man die Werte dieser beiden Speicherstellen mittels POKE ändert, kann man die Bildschirmdarstellung ganz einfach manipulieren (siehe auch »FARBEN EINSTELLEN« im Kapitel »Einführung«). Welche Werte welche Farben ergeben, können Sie in Anhang B erfahren.

Nachdem der Bildschirm gelöscht worden ist, wird eine ganze Zeile invertiert dargestellt (Zeile 1030) und danach eine weitere Zeile ganz gelöscht, sie »erscheint« in der Hintergrundfarbe. Mit der folgenden Schleife wird erreicht, daß jeweils am linken und rechten Bildschirmrand eine Spalte invers dargestellt wird. Der Bereich dazwischen ist leer. Für den unteren Abschluß wird noch einmal eine Zeile invers dargestellt.

Um das Programm bedienerfreundlich zu gestalten, wird am unteren Bildschirmrand noch eine Minianleitung eingeblendet. Diese beiden Zeilen werden auch wieder invers dargestellt, diesmal aber in einer anderen Farbe (nämlich Weiß). Ganz am Ende wird der Cursor noch an die [HOME]-Position gesetzt (Zeile 1090). In 1100 sehen Sie wieder einen Bekannten. Mit dem POKE-Befehl werden alle Tasten auf Wiederholfunktion geschaltet, so daß sie solange ein Zeichen ausgeben, wie sie gedrückt werden.

Oben habe ich erklärt, wie man die Bildschirmfarben einstellen kann. Wie das mit der Zeichenfarbe geht, wissen Sie schon lange ([CTRL]+[ZIFFERNTASTE] bzw. [CBM]+[ZIFFERNTASTE]). In 1110 sehen Sie in einer Stringvariablen (FA$ wie FArbe) alle möglichen Zeichenfarben nebeneinander. Diese Zeichen erscheinen, wenn Sie eine der eben aufgeführten Tastenkombinationen betätigen. Das Spektrum reicht von Schwarz über Grau, Blau, Grün und Rot bis hin zu Weiß. Alles in allem haben Sie 16 Farben zur Verfügung. (Übrigens haben Sie auch für die Bildschirmfarben sechzehn Farben; beginnend bei null können Sie alle Zahlen bis 15 verwenden. Bei höheren Zahlen wiederholen sich die Farben nur.) In den Variablen »IX« und »IY« sind die Koordinaten des Cursors definiert. Er steht zu Beginn der Eingabe an der Stelle 2/1. In den Variablen »X2« und »Y2« sind dieselben Werte noch einmal voreingestellt. Diese beiden sind eigentlich nur zum Rechnen und zu Sicherheitszwecken eingeführt worden. Sie werden öfters sehen, daß

zuerst mit »X2« oder »Y2« gerechnet wird, diese Werte erst überprüft werden, bevor der Cursor dann wirklich an diese Stelle gesetzt wird. In Zeile 1130 sehen Sie auch gleich, nach welchen Koordinaten sich der Cursor hauptsächlich richtet.

Was Sie auf dem Bildschirm als Cursor sehen, wird im Programm durch Zeile 1140 dargestellt. Und zwar wird lediglich die Position des Cursors mit einer bestimmten Farbe (nämlich türkis) »eingefärbt« (invers dargestellt). In den Variablen »FR$« und »FB$« (wie FaRbe bzw. FarBe) werden zwei weitere Farben definiert, die immer wieder gezielt für die Darstellung geändert werden. Im Moment ist »FR$« mehr oder weniger farblos, während mit »FB$« der Invers-Modus und die Farbe schwarz eingeschaltet werden. »FB$« wäre also geeignet, bestimmte Teile wieder zu löschen, denn alle Zeichen sind unsichtbar, wenn sie mit schwarzer Farbe auf schwarzen Hintergrund geschrieben werden.

In 1160 wartet das Programm auf eine Eingabe. Die folgenden Zeilen springen – je nach Eingabe – zu unterschiedlichen Zeilen. Sollte Ihre Eingabe nicht zu den »erwünschten« Buchstaben gehören, springt das Programm von 1230 wieder nach 1160.

Im nächsten Abschnitt stehen die verschiedenen Anweisungen, die gemäß Ihrer Eingabe angesprungen werden. Sollten Sie [G] für »GO« eingegeben haben, springt das Programm von 1170 nach 1300, wo es in 1310 auf ein RETURN trifft. Das veranlaßt das Programm, zurückzukehren und zu Zeile 2000 zu springen; da wird sich »Würmli« dann den Kopf zerbrechen.

War Ihre Eingabe ein [Z] für Zeichnen, dann wird lediglich die Zeichenfarbe »FR$« neu gesetzt. Anschließend verzweigt das Programm zu einer weiteren Abfrage, die auf eine Richtungseingabe wartet. Das gleiche passiert bei einem [P] für Position und bei einem [W] für Löschen.

Bei Neubeginn springt das Programm einfach noch einmal an den Anfang des Unterprogramms »Vorbereitungen«. Wollten Sie jedoch das Programm beenden, wird eine Unterroutine ab Zeile 4000 angesprungen; sie löscht die untersten zwei Zeilen, so daß die Kurzanleitung verschwindet. In diese freien Zeilen wird eine Sicherheitsabfrage ausgegeben. Falls Sie die Abfrage mit [J] für »Ja« beantworten, wird das Programm beENDet. Andernfalls werden die zwei Zeilen noch einmal gelöscht und das Programm springt zu Zeile 1080, damit die Anleitung ausgegeben werden kann, und Sie weiter herumtüfteln können.

In den drei oben genannten Fällen benötigt das Programm noch die Richtung, in die der Cursor bewegt werden soll. Dazu wurde ein eigenes kleines Teilstück entwickelt, das auf eine weitere Eingabe wartet. Dabei kann eine Richtung eingegeben werden (Zeilen 1620 bis 1650). Haben Sie aber einen anderen Buchstaben gedrückt, springt der Interpreter von 1660 nach 1170, um zu testen, ob eine der anderen zulässigen Tasten gedrückt wurde. Durch diese Taktik wird gewährleistet, daß – haben Sie einmal einen Programmpunkt wie Zeichnen, Position(ieren) oder Löschen angewählt – Sie immer wieder eine Richtung angeben können. Genausogut können Sie aber auch zu dem »Hauptmenü« zurückkehren, indem Sie eine der dort zugelassenen Tasten eintippen.

Zunächst müssen wir uns aber mit den Tasten beschäftigen. Wie gesagt werden zunächst diese »Richtungstasten« abgefragt, die da wären: [R] für **r**echts, [L] für **l**inks, [T] für **t**ief und [H] für **h**och. Die entsprechenden Abfragen sehen Sie in den Zeilen 1620 bis 1650. Die Auswertung Ihrer Eingabe wurde analog zu der obigen Auswertung programmiert. Das kann man ganz einfach am Listing erkennen.

In allen vier Teilen wird je nach Ihrer Eingabe zuerst die X- oder die Y-Koordinate um eins erhöht oder erniedrigt. Sie können den Cursor also nicht schräg über das Spielfeld bewegen. Sollte die Koordinate noch im erlaubten Bereich liegen (siehe jeweilige Abfrage in Zeilen 1710, 1760, 1810 und 1860), kann getrost zur Ausgabe verzweigt werden (ab 1900). Andernfalls muß die Koordinate noch berichtigt werden. In der Praxis merken Sie das, sobald Sie an den Rand des Spielfeldes stoßen. Der Cursor bewegt sich nicht mehr weiter, sondern bleibt am Rand »hängen«. Andernfalls ginge er ja über das Spielfeld hinaus.

Ab 1900 erfolgt die Ausgabe des Cursors an die neue Stelle. Zunächst muß die richtige Farbe eingeschaltet werden. Dafür dient »FR$«. Sollte »FR$« jedoch »durchsichtig« sein, muß auf »FB$« zurückgegriffen werden (siehe Zeile 1900). Daraufhin kann man die Schreibmarke (vom Computer) an die Stelle des Cursors (vom Programm) setzen. Da diese beiden Cursor nicht gleich sind, verwende ich ab sofort Schreibmarke für den des Computers, und Cursor für den des Programms.

Als Koordinaten werden die alten Daten verwendet (IX und IY). In 1920 wird diese Stelle erst einmal gelöscht. In Zeile 1930 wird »FB$« neu bestimmt. Das ist jetzt etwas Größeres. Ab der Speicherstelle 55296 steht (analog zu Adresse 1024) ein Speicher, der für die Farben der einzelnen Bildschirmadressen zuständig ist. Setzen Sie zum Beispiel ein [A] in die linke obere Bildschirmecke, bewegen Sie den Cursor einige Zeilen nach unten und geben hier ein:

```
POKE 55296,1
```

Sofort ist der Buchstabe weiß. Auch dieser Speicher umfaßt 1000 Stellen (25 Zeilen zu je 40 Zeichen). Wie das mit den Zeilen und Spalten funktioniert, habe ich Ihnen schon früher beschrieben. Wie ich Ihnen vorhin erklärt habe, gibt es 16 Farben mit den Kennziffern 0 bis 15. Bei höheren Zahlen wiederholen sich die Farben nur. Das »AND 15« blendet höhere Zahlen aus. Wie das genau funktioniert, kann ich an dieser Stelle nicht beschreiben. Aber sollten Sie einmal sogenannte Binärarithmetik betreiben, kommen solche Probleme auf Sie zu. Das Thema ist zwar nicht allzuschwer, aber es würde den Rahmen dieses Buches bei weitem sprengen.

Das Ergebnis dieser Berechnung soll für die Funktion MID$ verwendet werden. Nach diesen Berechnungen hat man die tatsächliche Farbenzahl »in Händen«, also von 0 bis 15. Die Funktion MID$ kann aber mit der Null nicht viel anfangen, darum muß zu jedem Ergebnis noch eins addiert werden.

Wenn Sie sich jetzt die Zeile 1930 abschließend anschauen, werden Sie feststellen, daß sie nichts anderes tut, als der Variablen »FB$« den Wert der Farbe zuzuweisen, den die neue

Speicherstelle für den Cursor hat. Nehmen wir an, Sie setzen den roten Cursor an eine schwarze Bildschirmstelle. Dann hat nach diesen Berechnungen »FB$« den Wert für die Farbe Schwarz.

Anschließend wird die Schreibmarke auf die neuen Cursor-Werte gesetzt, wo der Cursor in 1950 noch ausgegeben wird. Zu guter Letzt werden die Variablen »IX« und »IY« auf den neuesten Stand gebracht (Zeile 1960). Ein Sprung nach 1600 garantiert, daß Sie immer wieder Buchstaben für die Richtungen eintippen können.

Diese Ausführungen waren extrem theoretisch. Ich kann Ihnen nur empfehlen, sich mit dem Programm vertraut zu machen. Wenn Sie meinen Ausführungen nicht vertrauen oder sie nicht ganz verstehen, experimentieren Sie herum, lassen Sie die eine oder andere Programmzeile (oder beide) weg, fügen Sie neue ein. Lassen Sie sich Variableninhalte mit PRINT ausgeben...

Sobald Sie Ihr Labyrinth konstruiert haben, werden Sie irgendwann einmal [G] für GO drücken. Daraufhin tritt Würmli in Aktion. Seine erste Amtshandlung in Zeile 2030 besteht darin, die untersten zwei Bildschirmzeilen löschen zu lassen. Daraufhin gibt er seinem Benutzer eine kurze Mitteilung aus.

Seine Startkoordinaten werden auf 1/1 gesetzt. An dieser Stelle erscheint ein kleines Herz. Mit den Zeilen 2080 bis 2100 wird die Stelle, an der er vorher stand, gelöscht. Beim ersten Durchlauf, gibt es noch keine »alte« Stelle für Würmli. Zu dem Zeitpunkt sind in »IX« und »IY« noch die Cursor-Daten aus dem Editor enthalten. Sobald Sie diesen also verlassen, wird Ihr Cursor verschwinden. Und das erledigen diese Programmzeilen.

In diesem Bereich werden für Cursor-Berechnungen noch zwei weitere Variablen eingeführt, »X3« und »Y3«. Sie geben die Veränderungen von Koordinaten an, ähnlich der Variablen »DI« im Programm »Buchstabendreher«. Diese Zeile besagt demnach, daß zunächst die X-Koordinate um eins erhöht werden und die Y-Koordinate unverändert bleiben soll. Diese Veränderungen entsprechen der RIchtung (RI) vier.

In Zeile 2120 wird Würmli gleich verschoben. Dabei wird zu seinen alten Koordinaten X2/Y2 jeweils X3/Y3 addiert. Sollte Würmli mit diesen neuen Koordinaten bei einem der beiden Ausgänge gelandet sein, wird das Programm beendet. Dazu werden einfach seine Koordinaten überprüft. Stimmen sie mit denen der Ausgänge (rechts und links oben am Bildschirm) überein, ist er draußen!

Bei der nächsten IF-Abfrage wird geprüft, ob Würmli droht, aus dem Labyrinth zu laufen. Sollte das so ein, muß sofort die Richtung gewechselt werden; darum springt das Programm nach Zeile 2180. In den anderen Situationen wird zunächst die Farbe der Stelle untersucht, auf der Würmli sich setzen will. Diese Zeile 2160 darf Ihnen keine Probleme mehr bereiten. Ist die Farbe schwarz (FA=0), bedeutet das, daß das Feld frei ist. In einem solchen Fall kann Würmli direkt an der Stelle ausgegeben werden (Sprung nach 2230).

Ist das Feld nicht frei, merkt das der Computer an dem anderen Farbwert für das betreffende Feld. Daher muß auch hier die Richtung geändert werden, ist also gleichbedeutend mit einem Anstoßen an den Rand.

Die Richtung wird um eins verkleinert (Zeile 2180). Sollte sie dabei den kleinstmöglichen Wert unterschreiten, wird sie wieder auf den höchsten Wert zurückgesetzt. Den Zeilen 2190 bis 2220 können Sie entnehmen, welche Bedeutung jede Richtung für die einzelnen Koordinaten hat. Die Richtungsänderung muß auf alle Fälle gleich mit den Koordinaten verrechnet werden. Darum wird sofort nach Zeile 2120 verzweigt. Wie es dann weiter geht, wissen Sie schon.

In Zeile 2230 wird die Schreibmarke zunächst auf die alten Koordinaten »X2« und »Y2« gesetzt. An dieser Stelle steht auf dem Bildschirm (noch) das Herz für Würmli. Doch Zeile 2240 löscht dieses Herz. Erst nach dieser Zeile werden die Koordinaten X2 und Y2 aktualisiert. Und auf diese neuen Werte wird auch gleich das Herz für Würmli gesetzt (2250 bis 2270).

Anschließend wird die Zahl der »CLicks« (CL) um eins erhöht, damit man am Ende weiß, wie viele Schritte der Wurm gebraucht hat, um an sein Ziel zu gelangen. Damit die Richtung öfters mal gewechselt wird, wird am Ende »RI« prinzipiell um eins erhöht. Aber auch hier wird darauf geachtet, daß »RI« nur vernünftige Werte annimmt (Zeile 2280).

Und zu guter Letzt springt das Programm zu Zeile 2190. Diesmal werde ich auch über den letzten Teil noch ein paar Worte verlieren müssen. Das Programm bricht ab, wenn Würmli nach den Berechnungen das Ende erreicht hat. Das bedeutet aber, daß seine neue und letzte Position auf dem Bildschirm noch gar nicht dargestellt wurde. Das übernehmen die Zeilen 3030 und 3040. Genauer gesagt, löschen diese beiden Zeilen das Herz an der alten Position.

Eine kurze Schleife läßt das Herz am Ausgang mehrmals aufblinken. Wenn Ihnen das zu schnell geht, können Sie ja innerhalb der K-Schleife eine kurze PAusen-Schleife einfügen. Mit dem Aufruf des Unterprogramms ab Zeile 4000 werden die beiden untersten Bildschirmzeilen ein letztes Mal gelöscht. Sehen Sie sich bitte noch einmal die Abbruchbedingungen im Hauptteil an (Zeilen 2130 und 2140). Da sehen Sie, daß das Programm auch dann abgebrochen wird, wenn Würmli nicht bis ans Ende vorgedrungen ist, sondern wieder am Anfang steht. Und um diesem Umstand gerecht zu werden, wird bei »ENDE« noch einmal zwischen beiden Fällen unterschieden.

11 Word Scramble

Einführung

Viele unter Ihnen werden dieses Spiel als Brettspiel bereits kennen. Es handelt sich aber nicht um eine genaue »Übersetzung« des Originalspiels auf den Computer.

Bei unserer Version sucht der Computer aus 100 Wörtern eines heraus, vertauscht die Buchstaben und setzt Ihnen dann diesen Buchstabensalat vor. Sie müssen versuchen, das ursprüngliche Wort wieder herauszufinden.

Ich sagte, der Computer würde sich ein beliebiges Wort aus den 100 vorhandenen aussuchen. Das stimmt nicht ganz. Diese 100 Wörter sind in fünf Schwierigkeitsstufen zu je 20 Wörtern aufgeteilt. Sie selbst bestimmen zu Beginn des Spieles, aus welchem Bereich das Wort genommen werden soll. Somit können Sie sich allmählich steigern...

Damit das Ganze nicht zu einfach wird, wurde auch noch eine Zeitbegrenzung eingeführt. Sollten Sie allerdings merken, daß die Zeit einfach zu kurz bemessen ist, können Sie die Zeitspanne heraufsetzen oder den Countdown völlig weglassen. Wie das geht, wird in der Programmbeschreibung nach dem Spiel genau erklärt.

Aber nicht nur bezüglich dieser Zeitspanne können Sie das Programm Ihren Wünschen anpassen. Eine weitere Möglichkeit wäre zum Beispiel, den »Wortschatz« des Programms zu verbessern, sei es durch weitere Schwierigkeitsstufen, oder durch einfaches Erweitern des Grundwortschatzes. Auch auf diese Erweiterungen wird in der Beschreibung eingegangen. Ist das vielleicht eine Herausforderung für Sie als angehender Basic-Programmierer?

Listing zu Word Scramble

```
10 REM ********************
20 REM ***              ***
30 REM *** WORD SCRAMBLE ***
40 REM ***              ***
50 REM ********************
60 REM
70 GOSUB 1000
80 GOSUB 2000
90 GOSUB 3000
100 END
1000 REM
1010 REM *** VORBEREITUNGEN ***
1020 REM
1030 PRINT "{CLR,2DOWN}";TAB(13);"WORD SCRAMBLE"
1040 PRINT "{2DOWN}SIE KOENNEN WAEHLEN ZWISCHEN :"
1050 PRINT "{DOWN,5SPACE}1) SEHR LEICHT"
1060 PRINT "{5SPACE}2) LEICHT"
1070 PRINT "{5SPACE}3) MITTELSCHWER"
1080 PRINT "{5SPACE}4) SCHWER"
1090 PRINT "{5SPACE}5) SEHR SCHWER"
1100 INPUT "{2DOWN}IHRE WAHL (1-5) ";AN
1110 IF AN<1 OR AN>5 THEN GOTO 1100
1120 DIM WO$(100)
1130 FOR I=1 TO 100
1140 :READ WO$(I)
1150 NEXT I
1170 DATA SEE, UHR, WUT, MUT, IHR, WAND, SIE, EIS, BAD, LID
1180 DATA NAME, ROT, TOR, WEIL, ICH, TEE, GUT, TUER, DREI, LIEB
1185 REM ******************************
1190 DATA GRAS, BART, WIND, RUND, LIED, BLAU, PARK, TIEF, BALL, HAAR, HART
1200 DATA HOCH, AUGE, GRAU, MAUS, RAUS, RING, SEHR, BILD, SCHILD
1205 REM ******************************
1210 DATA FENSTER, GARTEN, STRAUSS, VOGEL, ZANGE, HAMMER, SCHRAUBE, PLATTE
1220 DATA WOLLE, KLEBER, SCHERE, FEUER, STIFT, KAKTUS, BLUME, LANZE
1230 DATA KATZE, PFEIFE, FEUER, STIFT
1235 REM ******************************
1240 DATA ORIGINAL, COMPUTER, FLOPPY, DISKETTE, SOFTWARE, HAUSHALT, MAGAZINE
1250 DATA PFLANZE, NATUR, SCHIRME, NAEHFADEN, TELEPHON, SCHACHT, ARMBAND
1260 DATA UEBERSCHRIFT, APFELMUS, KAESEKUCHEN, SYSTEM, AQUARELL, PERSONAL
1265 REM ******************************
1270 DATA KREISUMFANG, ATHLETEN, ENTHALTEN, TASCHENUHR, SCHATZTRUHE, BUT
     TERMILCH
1280 DATA BLUMENVASE, MECHANISMUS, PROJEKTION, WOLLKNAEUEL, KAMILLENTEE
1290 DATA COMPUTERPAPIER, ASESSOR, HEFTKLAMMER, TASTATUR, STRAUSSENEIER
1300 DATA UEBERWEISUNG, BANKROTT, DISKETTENHUELLE, HAUSSCHLUESSEL
1310 DIM WR$(15), WF$(15), WK$(15)
1320 RETURN
2000 REM
2010 REM *** SPIEL ***
2020 REM
2030 WO=(AN-1)*20+INT (RND(1)*20)+1
2040 WL=LEN (WO$(WO))
2050 FOR I=1 TO WL
2060 :WR$(I)=MID$(WO$(WO),I,1)
2070 :WF$(I)=WR$(I)
2080 NEXT I
2090 FOR I=1 TO WL
2100 :TX=INT (RND(1)*WL)+1
2110 :H$=WF$(I)
2120 :WF$(I)=WF$(TX)
2130 :WF$(TX)=H$
2140 NEXT I
```

```
2150 PRINT "{CLR,2DOWN}";TAB(13);"WORD SCRAMBLE"
2160 PRINT "{3DOWN}HIER IST IHR WORT :{2DOWN}"
2170 PRINT TAB(10);
2180 FOR I=1 TO WL
2190 :PRINT WF$(I);
2200 NEXT I
2210 PRINT:PRINT TAB(10);"{DOWN}";
2220 FOR I=1 TO WL
2230 :PRINT "-";
2240 NEXT I
2250 PRINT:PRINT "{4DOWN}BITTE EIN BUCHSTABE VON A BIS Z"
2260 PRINT "<RETURN>, WENN FERTIG ..."
2270 DG=DG+1
2280 FOR I=250*AN TO 1 STEP -1
2290 :IF PEEK(197)<>64 THEN GOTO 2350
2300 :IF INT(I/50)*50<>I THEN GOTO 2330
2310 :POKE 211,25:POKE 214,8:SYS 58640
2320 :PRINT "ZEIT :";INT (I/50);"{LEFT,2SPACE}"
2330 NEXT I
2340 RETURN
2350 POKE 211,9+DG:POKE 214,11:SYS 58640
2360 GET AN$:IF AN$="" THEN GOTO 2300
2370 IF ASC(AN$)=20 THEN GOTO 2460
2380 IF ASC(AN$)=21 THEN GOTO 2500
2390 IF ASC(AN$)=13 THEN RETURN
2400 IF AN$<"A" OR AN$>"Z" THEN AN$="-"
2410 WK$(DG)=AN$
2420 PRINT AN$;
2430 DG=DG+1
2440 IF DG>WL THEN DG=1
2450 GOTO 2300
2460 IF WK$(DG)="" THEN PRINT "-";:GOTO 2480
2470 PRINT WK$(DG)
2480 DG=DG-1:IF DG<1 THEN DG=WL
2490 GOTO 2300
2500 IF WK$(DG)="" THEN PRINT "-";:GOTO 2520
2510 PRINT WK$(DG);
2520 DG=DG+1:IF DG>WL THEN DG=1
2530 GOTO 2300
3000 REM
3010 REM *** ENDE ***
3020 REM
3030 FOR I=1 TO WL
3040 :POKE 211,9+I:POKE 214,11:SYS 58640
3050 :PRINT WR$(I);
3060 :IF WK$(I)=WR$(I) THEN WK=WK+1
3070 NEXT I
3080 IF WK<>WL THEN GOTO 3130
3090 PRINT:PRINT "{4DOWN}GRATULIERE !!{18SPACE}"
3100 PRINT "SIE HABEN DAS WORT ERRATEN !"
3120 GOTO 3160
3130 PRINT:PRINT "{4DOWN}DAS SPIEL IST BEENDET. ENDE DER";WL
3140 PRINT "BUCHSTABEN.{14SPACE,DOWN}"
3150 PRINT WK;"DAVON WURDEN ERREICHT."
3160 PRINT "{DOWN}MOECHTEN SIE NOCH EINMAL ?"
3170 GET AN$:IF AN$="" THEN GOTO 3170
3180 IF AN$="J" THEN RUN
3190 RETURN
```

Programmbeschreibung für Word Scramble

Bis zur Zeile 1110 sollte alles klar sein. Da kommt nichts Neues vor. Und die nächsten vier Zeilen bringen auch nichts Weltbewegendes. Schließlich wird ein ganz einfaches String-Array mit 100 Elementen definiert und anschließend alle Elemente mit einer READ-Anweisung belegt.

Vielleicht fragen Sie sich, wie viele Elemente ein solches Array maximal besitzen kann? Die Antwort darauf ist denkbar einfach: beliebig viele. Es gibt natürlich eine Einschränkung: Sie können nicht mehr Variablen anlegen als Speicher vorhanden ist. Aber prinzipiell gibt es hinsichtlich des C64-Basic keine Einschränkung. Sie können es ja einmal ausprobieren: Geben Sie einen DIM-Befehl nach dem anderen ein, bis der Computer den Fehler »?OUT OF MEMORY ERROR« ausgibt. Dimensionieren Sie beispielsweise ein Feld »A« mit 200 Elementen. Sie werden sehen, daß keine Fehlermeldung erscheint. Wenn Sie jetzt dieselbe Zeile noch einmal eingeben mit – meinetwegen – 400 Elementen, bekommen Sie eine Fehlermeldung, allerdings eine andere als oben angesprochen. In diesem Fall erhalten Sie »?REDIM'D ARRAY ERROR«. An dieser Stelle kann ich also gleich einen neuen Befehl einfließen lassen, den CLR-Befehl (für CLeaR). Sobald Sie alle Variableninhalte löschen wollen, verwenden Sie den CLR-Befehl. Aber passen Sie auf: Es wird keine Variable davon ausgenommen.

Sollten Sie ein Array neu dimensionieren wollen, um es (wie in unserem Fall) zu vergrößern, müssen Sie vorher den CLR-Befehl anwenden, sonst kommt es zur oben genannten Fehlermeldung. Aber Sie verlieren alle Variableninhalte!

Sie werden bei Ihren Tests feststellen, daß Sie bis zu 7779 Variablen dimensionieren können, wenn Sie sonst nichts im Speicher haben. Das gilt aber nur für normale Zahlenvariablen, zum Beispiel: DIM A(7779).

Die Gelegenheit ist günstig, einen neuen Variablentyp einzuführen: die INTeger-Variable. Sie haben ja schon einige Male mit dem INT-Befehl gearbeitet und wissen, daß er die Nachkommastellen einer Zahl abschneidet. Und wie nicht anders zu erwarten, sind INTeger-Zahlen nur ganze Zahlen, also ohne Nachkommateil. Sie werden recht selten gebraucht. Wenn man sie benützt, werden sie durch ein nachgestelltes Prozentzeichen kenntlich gemacht, zum Beispiel »A%«. Man kann aber die »normalen« Zahlenvariablen genausogut hernehmen; und da diese normalen Variablen im C64-Basic etwas schneller arbeiten als ihre INTeger-Kollegen, werden sie prinzipiell bevorzugt.

Kommen wir nach diesen allgemeinen Ausführungen aber wieder zu unserem Programm zurück: Wie Sie den Zeilen 1050 bis 1090 entnehmen können, werden die zu erratenden Wörter in fünf Schwierigkeitsgrade unterteilt. Insgesamt stehen 100 Wörter zur Verfügung; damit es gerecht ist, hat jede Gruppe 20 Wörter als Grundlage. Sie sehen diesen »Wortschatz« in den Zeilen 1170 bis 1300. Wegen der besseren Untergliederung wurden REM-Zeilen zwischen diese DATA-Wüsten eingeschoben. Man könnte sonst keinen Überblick mehr behalten.

Im Anschluß an diesen Wortschatz finden Sie noch drei Dimensionierungen. Die 15 steht dabei für die höchste Buchstabenzahl eines Wortes. Auf gut deutsch: Das längste Wort unter diesen 100 Wörtern hat 15 Buchstaben. Den Abschluß bildet – wie könnte es anders sein? – ein RETURN.

Ich möchte noch einen kleinen Tip geben: Sie können die Zeilen 1310 und 1320 noch vor die ganzen DATA-Listen stellen. Dann dauern die Vorbereitungen nicht so lange. Und daran können Sie noch etwas sehen: Der DATA-Befehl ist der einzige, an den der Interpreter nicht direkt »kommen« muß. Auch wenn das Programm noch vor den DATA-Zeilen wieder zurückkehrt, werden die Daten mit dem READ-Befehl ordnungsgemäß eingelesen.

In Zeile 2030 wird mit der Zufallsvariablen »WO« ein WOrt aus den 100 möglichen ausgesucht. Man darf die Wortwahl natürlich nicht nur dem Computer überlassen. Man muß auch die Wahl des Benutzers berücksichtigen. Dadurch kommt es zu dieser »komischen« Zeile 2030. Mit »(AN–1)*20« wird der Bereich angewählt, der vom Benutzer ausgesucht wurde. Und die Zufallsfunktion sucht sich dann aus diesem Bereich eines von 20 Wörtern heraus.

In der nächsten Zeile lernen Sie mal wieder einen neuen Befehl kennen: Mit der Funktion »LEN« können Sie die Länge eines Strings erfahren. Da in »WO$( )« alle 100 Wörter gespeichert sind, kann man mit »WO$(WO)« ganz einfach auf das ausgesuchte Wort zugreifen. Mit LEN für LENgth (englisch, Länge) erfährt man, wie viele Buchstaben das Wort enthält.

In der Schleife ab Zeile 2050 wird dem Array »WR$« (Wort Richtig) jeder einzelne Buchstabe des Wortes zugeordnet. Mit der bekannten Funktion »MID$« kann man gezielt auf die Buchstaben zugreifen. Sie sehen, wie bequem man programmieren kann, wenn man die nötigen Funktionen zur Verfügung hat und auch kennt!

Der Variablen »WF$« (Wort Falsch) wird noch einmal derselbe Wert zugewiesen wie »WR$«, so daß am Ende der Schleife »WF$« eine genaue Kopie von »WR$« darstellt. Aber mit der nächsten Schleife wird das wieder beendet. Da werden die Buchstaben in »WF$« nämlich zu einem Buchstabensalat verrührt, den Sie wieder entwirren sollen. Dazu wird der Variablen »TX« ein Zufallswert zwischen eins und der Wortlänge zugewiesen. Einer Hilfsvariablen »H$« wird der Wert »WF$(I)« – also nacheinander alle Buchstaben des Wortes – zugewiesen. Dadurch wird ein »Dreieckstausch« zwischen »WF$(I)« und »WF$(TX)« mit Hilfe von »H$« möglich.

Nach dieser zweiten Schleife steht also in »WR$« das richtige Wort, während »WF$« alle Buchstaben verdreht enthält. Die nächsten drei Zeilen geben eine Meldung auf dem Bildschirm aus und setzen den Cursor abschließend noch in die Spalte zehn. Hier beginnt eine Schleife, die alle Elemente von »WF$« nebeneinander (wegen des Strichpunktes) auf dem Bildschirm ausgibt. Nach dieser Schleife haben Sie praktisch einen Buchstabensalat auf dem Bildschirm, der – richtig sortiert – ein sinnvolles Wort ergibt.

Zeile 2210 gibt zwei Zeilen weiter unten auf dem Bildschirm so viele »–« aus, wie das Wort Buchstaben hat, pro Buchstaben einen Strich.

Weitere vier Zeilen weiter unten gibt die Zeile 2250 eine Aufforderung aus.

In Zeile 2270 wird noch die bekannte Variable »DG« für DurchGang um eins erhöht. Diesmal ist diese Variable allerdings nicht dafür da, Ihre Versuche zu zählen, sondern sie soll mitzählen, für welchen Buchstaben Sie gerade eine Eingabe tätigen. Und zwar sollen Sie doch einen Buchstaben nach dem anderen eingeben; da muß das Programm wissen, ob Sie bereits am Ende des Wortes angekommen sind, bzw. welcher Stelle es Ihrer Eingabe überhaupt zuordnen soll. Es dient also unter anderem auch als Index (siehe auch Zeile 2410). In diesem speziellen Fall wird die Variable aber nur auf eins gesetzt, da das Programm zum ersten Mal auf diese Variable stößt und später nicht mehr auf diese Zeile treffen wird.

Die nächste Schleife stellt eine zeitliche Begrenzung für Ihre Eingabe dar. Und zwar wird von einem Wert bis eins gezählt; bei der Gelegenheit sehen Sie auch gleich, wie man eine Schleife mittels »STEP −1« rückwärts laufen lassen kann. »STEP« können Sie ganz allgemein verwenden, wenn Sie die Zählvariable einer Schleife nicht um eins erhöhen wollen. Beispiel: FOR A=1 TO 10 STEP 0.01.

Die verbleibende Zeit wird in Sekundenabständen dargestellt. Zunächst wird in der Schleife geprüft, ob Sie irgendeine Taste gedrückt haben. Der Inhalt der Speicherstelle 197 ist 64, solange keine Taste gedrückt wurde. Wurde *keine* Taste gedrückt, springt der C64 in die Zeile 2300. Dort kommt gleich die nächste Abfrage. Ist der Schleifenzähler ein Vielfaches von 50, sollen die Befehle in 2310 und 2320 ausgeführt werden. Sie können ganz einfach überprüfen, ob eine Variable Vielfache einer Zahl ist, indem Sie die Variable durch die Zahl dividieren, die Nachkommastellen mit »INT« abschneiden und dieses Ergebnis noch einmal mit der Zahl multiplizieren. Dieses Ergebnis müssen Sie mit der ursprünglichen Variablen vergleichen. Sind beide Zahlenwerte gleich, dann handelt es sich um ein Vielfaches.

Ist der Zähler kein Vielfaches von 50, springt der Computer zu dem NEXT in 2330. Gehen wir das Ganze noch einmal an einem praktischen Beispiel durch: Nehmen wir an, Sie drücken keine Taste und haben den Schwierigkeitsgrad 2 gewählt. Die Schleife beginnt bei 500 (»AN« hat immer noch den Schwierigkeitsgrad gespeichert!) und zählt bis 1. Da Sie keine Taste gedrückt haben, ist »PEEK(197)<>64«, die Bedingung in Zeile 2290, also nicht erfüllt (»<>« steht für ungleich im Gegensatz zu »=«). Gehen wir weiter davon aus, daß die Schleife zum ersten Mal durchlaufen wird, »I« also den Wert 500 enthält. I/50 ergibt zehn, INT verändert das Ergebnis nicht, mal 50 ergibt wieder 500. Das bedeutet also, die Bedingung ist nicht erfüllt, der Computer kommt zu Zeile 2320, er gibt Ihnen die verbleibende Zeit aus.

Haben Sie mittlerweile eine Taste gedrückt, ist die Bedingung für 2290 erfüllt, der Interpreter springt zu Zeile 2350. Die Variable »DG« »weiß«, für welche Stelle Sie gerade einen Buchstaben eingegeben haben. »DG« kann Werte zwischen eins und der Wortlänge annehmen; daher kann man es auch direkt für die Bildschirmsteuerung einsetzen, wie Zeile 2350 zeigt. Der Cursor wird an die richtige Stelle gesetzt, an der später Ihre Eingabe angezeigt werden soll.

Daraufhin muß dem Programm erst einmal mitgeteilt werden, welche Taste Sie betätigt haben. Schließlich »weiß« es erst, daß ein Tastendruck erfolgt ist. Das geht ganz leicht mit dem GET-Befehl. Dazu muß man wissen, daß alle Tastendrücke erst einmal im sogenannten Tastaturpuffer zwischengespeichert werden, bevor Sie als Basic-Programmierer Zugriff dazu haben. Sobald dieser Puffer sich füllt, wird der Inhalt der Speicherstelle 197 geändert.

Die ASC-Funktion haben Sie bereits im vorletzten Spiel kennengelernt. Hier wird sie benutzt, um die Codes der einzelnen Tasten abfragen zu können. Wenn Sie sich dazu Anhang F Ihres Commodore-Handbuches ansehen, können Sie jeder Taste einen ASCII-Code zuordnen. Somit können Sie ganz einfach die entsprechenden Tasten abfragen. Eine andere Möglichkeit wäre natürlich, die Zeichen in Anführungszeichen zu setzen und so abzufragen. Da es sich bei diesen Tasten aber um sogenannte Sondertasten handelt, stünden im Programmtext noch mehr inverse Zeichen (in dem Fall für [INST/DELETE] und [CRSR-RIGHT]). Aber selbst wenn man das in Kauf nähme, erklären Sie mir bitte, wie Sie ein inverses Zeichen für [RETURN] auf den Bildschirm bringen können? Bei der Abfrage in Zeile 2390 handelt es sich nämlich um die [RETURN]-Taste, und dies ist die einzige Möglichkeit, sie abzufragen. Und wenn man eine Taste mit dem ASCII-Code abfragen muß, kann man das doch auch für die anderen Tasten so programmieren.

In den Zeilen 2370 bis 2390 werden also die Sondertasten abgefragt. Zum einen gibt Ihnen das die Möglichkeit, mit [INST/DEL] bei der Eingabe um einen Buchstaben nach links, bzw. mit der [CRSR-RIGHT]-Taste um eins nach rechts zu »springen«. Damit wird diese Eingaberoutine etwas flexibler gestaltet. Erst wenn Sie [RETURN] drücken, beginnt das Programm mit der Auswertung Ihrer Eingabe; und die beginnt ab Zeile 3000. Doch dazu später.

In Zeile 2400 wird Ihre Eingabe auf zulässige Buchstaben hin überprüft. Zahlen und Sonderzeichen müssen ausgeschlossen werden. War Ihre Eingabe »unzulässig«, wird für die Bildschirmausgabe ein »-« vorgesehen. Andernfalls wird der Variablen »WK$(DG)« (in der Annahme, Ihr Raten ist erfolgreich: Wort Korrekt) Ihre Eingabe zugewiesen (Zeile 2410). In 2420 wird diese Eingabe auch noch ausgegeben. Damit Ihre nächste Eingabe um eine Stelle weiter rechts erfolgt, muß »DG« um eins erhöht werden (siehe auch Zeile 2350).

Wie weiter oben bereits erläutert, zeigt »DG« immer auf eine Stelle im Wort; an diese Stelle muß Ihre Eingabe gesetzt werden. Aber was soll passieren, wenn Sie bereits so viele Buchstaben eingegeben haben, wie das Wort hat? Soll man dann die weitere Eingabe abblocken? Oder die Eingabe beenden?

Dieses Programm bietet (wie ich meine) eine komfortablere Möglichkeit: Sobald Sie das Wortende erreichen, springt es wieder an den Wortanfang. Das gleiche gilt umgekehrt auch: Wenn Sie mit [INST/DEL] an den Wortanfang gekommen sind, setzt das Programm die Schreibmarke automatisch wieder auf den letzten Buchstaben.

Dafür ist die Abfrage in Zeile 2440 zuständig. Wird DG größer als die Wortlänge, wird DG wieder auf den ersten Buchstaben gesetzt. Anschließend springt das Programm wieder in die Zeitschleife, um zu überprüfen, ob Ihre Zeit bereits abgelaufen ist.

Ab Zeile 2460 steht die Sonderbehandlung für die [INST/DEL]-Taste. Sollte für den Buchstaben »DG« noch keine Eingabe erfolgt sein (WK$(DG)=""), dann wird ein »-« ausgegeben, andernfalls die Eingabe selbst (Zeilen 2460 und 2470). Dazu müssen Sie beachten, daß in »WK$« auch Eingaben gespeichert sind, die Sie etwas früher getätigt haben. Stellen Sie sich vor, Sie drücken so lange [CRSR-RIGHT] bzw. [INST/DEL], bis Sie mehrmals das Wort »umrundet« haben. Sollten Sie davor Eingaben getätigt haben, sind diese alle noch in »WK$« gespeichert. Daher diese Abfrage.

Da bei der [INST/DEL]-Taste der Cursor um eins nach links gesetzt werden soll, muß »DG« um eins verringert werden. Hier kommt wieder die Abfrage, ob bereits der Wortanfang erreicht wurde. Wenn ja, muß »DG« auf die Wortlänge »WL« gesetzt werden (Zeile 2480). Anschließend erfolgt auch von hier wieder ein Sprung in die Zeitschleife.

Ab Zeile 2500 steht noch die Sonderbehandlung für die [CRSR-RIGHT]-Taste. Sie entspricht der eben besprochenen für [INST/DEL]. Im letzten Abschnitt erfolgt die Überprüfung Ihrer Eingabe und deren Auswertung.

Zunächst jedoch werden Ihre Buchstaben mit der richtigen Lösung überschrieben: Zeilen 3030 bis 3070. Dazu verwendet das Programm die Variable »WR$«. Was in dieser Routine die Schleifenvariable »I« verrichtet, war im Programmteil »SPIEL« die Variable »DG«.

Bei jedem Durchgang wird noch gleich geprüft, ob Ihr eingegebener Buchstabe mit dem richtigen übereinstimmt (Zeile 3060). Sind sie gleich, wird die Variable »WK« (Wort Korrekt) um eins erhöht. Um Mißverständnissen vorzubeugen: Der Computer unterscheidet zwischen »WK« und »WK$«, er kann die beiden also nicht miteinander verwechseln.

Nach der Schleife muß dann lediglich überprüft werden, wie viele richtige Buchstaben Sie hatten. Stimmt diese Zahl mit der Wortlänge überein (Zeile 3080), dann haben Sie das Wort richtig erraten. Andernfalls wird auf dem Bildschirm ausgegeben, wie viele Buchstaben Sie herausgefunden haben.

12 Prim-Man

Einführung

Eigentlich hat dieses Programm mit Primzahlen wenig am Hut. Aber es spielt mit Zahlen, und Primzahlen kommen bei diesen Zahlen schon auch vor. In diesem Spiel erstellt der Computer eine Liste aller Zahlen von 1 bis zu einer Zahl, die Sie ihm vorher eingegeben haben. Sie werden nun aufgefordert, eine beliebige Zahl aus dieser Liste zu nehmen. Ihr »Kontostand« wird um den Wert dieser Zahl erhöht.

Der Computer zerlegt Ihre Zahl in alle ihre Faktoren (also auch in die Primfaktoren (!)) und entfernt diese dann aus der Liste. Ich werde Ihnen das später an einem Beispiel verdeutlichen.

Ziel des Spieles ist es natürlich, möglichst viele Punkte auf Ihrem Punktekonto anzusammeln. Allerdings hat der Computer einige Vorteile: Zum Beispiel gibt es da die Regel, daß Sie nur Zahlen entnehmen dürfen, von denen noch mindestens ein Faktor in der Liste enthalten ist. Primzahlen und solche Zahlen, deren Faktoren bereits entfernt wurden, sind demnach ausgenommen (Primzahlen sind solche Zahlen, die nur durch eins und sich selbst teilbar sind, zum Beispiel 2, 3, 5, 7, 11 etc.)

In der Praxis bedeutet das, daß am Ende ziemlich viele Zahlen übrigbleiben, die Sie gemäß obiger Regel nicht verwenden dürfen. Alle übriggebliebenen Zahlen werden aber dem Computer gutgeschrieben. Und das verschlechtert Ihre Aussichten auf einen Sieg ganz erheblich.

Damit wäre das Spielprinzip bereits erklärt. Aber ich möchte Ihnen an einem Beispiel noch die Funktionsweise genauer erläutern. Nehmen wir als Beispiel eine Liste der Zahlen von 1 bis 12:

1 2 3 4 5 6 7 8 9 10 11 12

Nehmen wir an, Sie wählen die zwölf:

Der Computer zerlegt sie in alle mögliche Faktoren (nicht nur die Primfaktoren):

1, 2, 3, 4, 6 und 12, da $1*12 = 2*6 = 3*4 =12$

Die 12 selbst haben Sie bereits erhalten, daher bekommt der C64

$1 + 2 + 3 + 4 + 6 = 16$,

der Computer hat bereits einen Vorsprung. Die Liste enthält jetzt noch folgende Zahlen:

5 7 8 9 10 11

Die einzige Zahl, die Sie jetzt noch wählen können, ist die zehn. Die anderen Zahlen sind Primzahlen (5, 7, 11), oder es gibt keinen Faktor mehr in der Liste. Also wählen Sie die zehn. Im folgenden Schritt merkt der Computer, daß Sie keine Zahl mehr auswählen können, addiert alles auf, und kommt zu dem überraschenden Ergebnis:

Computer: 56
Sie: 22

Soll ich Ihnen zeigen, wie Sie hätten gewinnen können? Soweit will ich jetzt gar nicht gehen, aber ein Unentschieden können Sie mit der Reihenfolge 11, 6, 12, 10 erreichen; probieren Sie es aus!

Listing zu Prim-Man

```
10 REM ****************
20 REM ***          ***
30 REM *** PRIM-MAN ***
40 REM ***          ***
50 REM ****************
60 REM
70 GOSUB 1000
80 GOSUB 2000
90 GOSUB 3000
100 END
1000 REM
1020 REM *** VORBEREITUNG ***
1030 DIM LI(50)
1040 FOR I=1 TO 50
1050 :LI(I)=I
1060 NEXT I
1070 PRINT "{CLR,2DOWN}WIE VIELE ZAHLEN SOLL DIE LISTE ENTHAL-"
1080 INPUT "TEN (1-50) ";AN
1090 IF AN<1 OR AN>50 THEN GOTO 1070
1100 NU=AN
1110 RETURN
2000 REM
2010 REM *** SPIEL ***
2020 REM
2030 PRINT "{CLR,2DOWN}";TAB(12);"PRIM-MAN"
2040 PRINT "{2DOWN}HIER IST DIE LISTE:"
2050 PRINT
2060 FOR I=1 TO NU
2070 :IF LI(I)=0 THEN PRINT "{3SPACE}";
2080 :IF LI(I)<10 THEN PRINT " ";
2090 :IF LI(I) THEN PRINT I;
2100 NEXT I
2110 IF NU=1 THEN GOTO 2480
2120 FOR I=2 TO NU
2130 :IF LI(I)=0 THEN GOTO 2460
2140 :FOR J=1 TO I
2150 : IF LI(J)=0 THEN GOTO 2450
2160 : IF J=I THEN GOTO 2450
2170 : IF LI(I)/J<>INT (LI(I)/J) THEN GOTO 2450
2180 : POKE 211,0:POKE 214,13:SYS 58640
2190 : PRINT "DIE BETRAEGE :{2SPACE}COMPUTER:";CO
2200 : PRINT "{21SPACE}SIE:";SI
2210 : AN=0
2220 : INPUT "{DOWN}WELCHE ZAHL WOLLEN SIE ";AN
2230 : IF AN<1 OR AN>NU THEN GOTO 2220
2240 : IF LI(AN) AND AN=INT(AN) THEN GOTO 2270
2250 : PRINT "DIESE ZAHL IST NICHT MOEGLICH !{2UP}"
2260 : GOTO 2220
2270 : SC=0
2280 : IF AN=1 THEN GOTO 2330
2290 : FOR K=1 TO AN
2300 :  IF LI(K)=0 THEN GOTO 2325
2310 :  IF K=AN THEN GOTO 2325
2320 :  IF AN/K=INT(AN/K) THEN SC=SC+K
2325 : NEXT K
2330 : IF SC<>0 THEN GOTO 2370
2340 : PRINT "GEHT LEIDER NICHT."
2350 : PRINT "FUER MICH MUSS ETWAS UEBRIGBLEIBEN."
2360 : GOTO 2220
2370 : LI (AN)=0
2380 : SI=SI+AN
2390 : CO=CO+SC
```

```
2400 : FOR K=1 TO AN
2410 :   IF LI(K)=0 THEN GOTO 2430
2420 :   IF AN/K=INT(AN/K) THEN LI(K)=0
2430 : NEXT K
2440 : GOTO 2030
2450 :NEXT J
2460 NEXT I
2470 RETURN
2480 PRINT "{2DOWN}SIE KOENNEN NICHTS MEHR NEHMEN ..."
2490 PRINT "{DOWN}WEITER MIT <RETURN> !"
2500 GET A$
2510 IF A$<>CHR$(13) THEN GOTO 2500
2520 LI(1)=0
2530 RETURN
3000 REM
3010 REM *** ENDE ***
3020 REM
3030 PRINT "{CLR,6DOWN}DAS SPIEL IST BEENDET !"
3040 CO=0
3050 FOR I=1 TO NU
3060 :CO=CO+I
3070 NEXT I
3080 CO=CO-SI
3090 PRINT "{DOWN,5SPACE}COMPUTER:";CO
3100 PRINT "{10SPACE}SIE:";SI
3110 PRINT "========================="
3120 IF CO>SI THEN PRINT "ICH HABE GEWONNEN !"
3130 IF CO<SI THEN PRINT "SIE HABEN MICH BESIEGT !"
3140 IF CO=SI THEN PRINT "SIE HABEN EIN UNENTSCHIEDEN GESCHAFFT !"
3150 RETURN
```

■ Programmbeschreibung für Prim-Man

Als erste »Vorbereitung« für das Spiel wird eine Variable »LI« (wie LIste) dimensioniert und mit Werten vorbelegt. Im Anschluß daran werden Sie gefragt, wie groß diese Liste werden soll. Dabei können Sie zwischen eins und 50 wählen. Diese Abfragezeile erledigt 1090. Anschließend wird noch die Variable »NU« (wie NUmmer) auf denselben Wert wie »AN« (Ihre ANtwort) gesetzt.

Man könnte das Programm freilich etwas flexibler gestalten, indem man das Feld »LI« erst nach Ihrer Eingabe genau mit diesem Wert dimensioniert. Dadurch könnte man Speicherplatz und Zeit sparen. Aber das fällt bei diesem Programm überhaupt nicht ins Gewicht. Nichtsdestoweniger könnten Sie das vielleicht einmal machen! Es sollte Sie vor keine unlösbare Aufgabe stellen, die Reihenfolge der Zeilen etwas umzustellen und als obere Begrenzung nicht 50, sondern ... herzunehmen!

Kommen wir zum Hauptteil: Die ersten drei Zeilen *müssen* klar sein, Sie haben gar keine andere Wahl! Danach soll mit einer kleinen Schleife die LIste auf dem Bildschirm ausgegeben werden, damit Sie darüber informiert werden, wie viele Zahlen sie (noch) enthält. Innerhalb der Schleife werden verschiedene Fälle unterschieden: Falls LI=0 ist, wird eine größere Lücke mit Hilfe von »SPACEs« dargestellt. Ist die Zahl kleiner als zehn, wird erst mal nur ein »SPACE« dargestellt. Jetzt kommen wir aber zu einer etwas merkwürdigen Abfrage: IF LI(I) THEN ...

Das Rätsel läßt sich ganz einfach lösen: Diese Abfrage ist für alle »LI« erfüllt, *außer* für »LI=0«. Gehen wir diese Abfrage also für drei unterschiedliche LI(I)-Werte durch, nämlich für null, fünf und zwölf.

Für die Null ist die erste Abfrage erfüllt. Es werden drei »SPACEs« ausgegeben. Die nächste Abfrage ist auch erfüllt; niemand wird abstreiten können, daß null kleiner als zehn ist. Demnach wird noch ein weiteres »SPACE« ausgegeben. Erst die letzte Abfrage versagt bei null.

Nehmen wir an, LI(I) wäre gleich fünf. Die erste IF-Abfrage ist offensichtlich nicht erfüllt; die zweite ist sicherlich richtig, also gibt es ein »SPACE«. Und die dritte gilt bekanntlich für alle Zahlen außer null, daher wird die Zahl selbst auch noch auf dem Bildschirm dargestellt. Für »LI(I)=12« läuft das Ganze so ähnlich; lediglich Zeile 2080 kann jetzt nicht mehr als erfüllt angesehen werden; darum gibt das Programm keinerlei [SPACE]s mehr aus.

Durch dieses Verfahren wird gewährleistet, daß die Zahlen immer recht ordentlich auf dem Bildschirm ausgegeben werden. Ansonsten wäre die Ausgabe zu unübersichtlich. Sollte die obere Grenze NU=1 sein, verzweigt das Programm nach 2480; das bedeutet nämlich, daß Sie keine weiteren Zahlen mehr nehmen können, und das Programm beendet werden muß.

Ansonsten beginnt in Zeile 2120 eine Schleife von zwei bis »NU«. Sollte das gerade angesprochene Feldelement LI(I) gleich null sein, springt das Programm zum NEXT I-Befehl (Zeile 2130). Innerhalb der I-Schleife gibt es noch eine weitere Schleife mit der Zählvariable »J«. Diese Schleife läuft vom Anfang bis zur Zählvariablen »I« der äußeren Schleife. Sollte das Element LI(J) null sein, verzweigt der Interpreter zum NEXT J-Befehl.

Sollte »J« gleich »I« sein, umgeht das Programm auch die gesamte Schleife und springt direkt zum NEXT J-Befehl. In der nächsten Zeile sehen Sie wieder eine Abfrage, ob eine Variable das Vielfache einer anderen ist. Diesmal wurde die Abfrage jedoch ein wenig umgestellt (öfter mal was Neues, oder auf (Computer-)lateinisch: Variatio delectat). Diesmal wird eine Variable durch eine andere dividiert und dann mit dem Ergebnis derselben Division ohne Nachkommastellen verglichen. Mit dieser Abfrage soll getestet werden, ob Sie überhaupt noch Zahlen aus der Liste nehmen können. Wenn das Programm nämlich feststellt, daß es zu keinem »LI(I)« mehr irgendwelche Faktoren »J« gibt, dann springt es immer von dieser Abfrage zu dem NEXT-Befehl in Zeile 2450. Irgendwann ist der einmal »am Ende«, dann kommt gleich der nächste NEXT-Befehl. Gibt es für gar kein »LI(I)« mehr irgendwelche Faktoren, trifft das Programm auf die Zeile 2470 und kehrt zurück.

Sobald das Programm aber Zeile 2180 abarbeitet, heißt das, daß es noch irgendeinen Faktor in der Liste gibt. Und in dem Fall können Sie nach Ihrer Wahl gefragt werden.

Zuvor wird jedoch der Kontostand ausgegeben. Nach Ihrer Wahl wird die Eingabe erst einmal überprüft (Zeilen 2230 und 2240). Zeile 2230 überprüft, ob Ihre Antwort überhaupt für diese Liste zutrifft. Zeile 2240 testet daraufhin, ob das Element mit Ihrer Eingabe (also LI(AN)) überhaupt existiert und ob Ihre Eingabe ganzzahlig ist (eine Zahl mit Nachkommastellen wäre zwar sinnlos, aber vielleicht haben Sie sich ja vertippt?). Nur wenn beide

Bedingungen erfüllt sind (wegen des AND), springt der Computer an die Zeile 2270. In den anderen Fällen wird eine Fehlermeldung ausgegeben, und Sie müssen Ihre Eingabe wiederholen.

In »SC« wird später gespeichert, wie viele Punkte der Computer nach Ihrer Eingabe sammeln konnte. Darum muß es zu Beginn jedes Durchlaufs wieder auf Null gesetzt werden. War Ihre Eingabe gleich eins, kann für den Computer nichts übrigbleiben. Darum kann er gleich zu Zeile 2330 springen, und dort wird er auf jeden Fall mit »SC=0« ankommen.

Sonst beginnt jetzt erst eine Schleife, die alle Elemente von Anfang bis zu Ihrer Antwort durchläuft. Ist ein Element null, kann der Computer gleich zum NEXT-Befehl springen. Die Null bedeutet nämlich, daß das Element bereits aus der Liste gestrichen wurde. Sollte »K« gleich Ihrer Antwort sein (beim letzten Durchlauf), wird auch gleich zum NEXT-Befehl verzweigt, da Sie diese Zahl gutgeschrieben bekommen, und nicht der Computer. Die übriggebliebenen Elemente werden noch daraufhin untersucht, ob Ihre Antwort »AN« ein Vielfaches des momentanen Schleifenzählers ist. Zum Schluß wird die Schleife noch geschlossen.

Gehen wir diese Schleife wieder an einem Beispiel durch: Angenommen, Sie geben bei Ihrer Wahl sechs ein, und es wurde noch kein Element aus der Liste entfernt. Dann läuft »K« von eins bis sechs. LI(K)=0 ist – nach den Voraussetzungen – nie erfüllt, das können wir also beiseite lassen.

»K=AN« ist nur für den letzten Schleifendurchlauf erfüllt, am Anfang können wir uns also voll auf Zeile 2320 konzentrieren. AN/K ist beim ersten Schleifendurchlauf 6/1, ergibt sechs, mit und ohne Nachkommastellen. Darum wird »SC« um eins erhöht.

Für »K=2« ergibt die Division wieder etwas Ganzzahliges, nämlich drei. Also beträgt »SC« mittlerweile drei (»SC« war eins, wird um »K«, sprich zwei, erhöht).

Bei »K=3« haben wir wieder denselben Fall, »SC« ist mittlerweile also sechs. Bei »K=4« und »K=5« geht die Division nicht auf, das heißt »AN« ist in dem Fall kein Vielfaches von »K«. Bei »K=6« ist die Bedingung in Zeile 2310 erfüllt, der Computer springt also zum NEXT-Befehl; da »K« aber nicht mehr weiter erhöht werden soll, bearbeitet der Interpreter die nächste Zeile (Zeile 2330).

In dieser Zeile wird überprüft, wie viele Punkte der Computer einheimsen konnte. Sind es mehr als null Punkte, dann geht Ihre Eingabe in Ordnung, andernfalls muß er leider Ihre Antwort aus Zeile 2220 zurückweisen, da er keine Faktoren Ihrer Eingabe finden konnte. Das ist aber eine Bedingung, die an Ihre Antwort gestellt wird.

Kann Ihre Antwort aber bestehen bleiben, dann wird als erstes das Element LI(AN) auf null gesetzt, sprich, dieses Element wurde aus der Liste genommen. Daraufhin wird Ihrem Kontostand (»SI« für SIe) der Wert AN gutgeschrieben. Und zwar wissen Sie ja noch aus den Vorbereitungen, daß jedes Element als Wert auch seinen Index hat (siehe Zeile 1050: LI(I)=I). Dem Computer werden die berechneten Punkte »SC« (für englisch SCore) zugerechnet. Anschließend müssen aber noch alle Elemente, die der Computer für sich in

Anspruch nahm, aus der Liste entfernt werden. Und die Bedingung für den Computer war, daß »AN« ein Vielfaches von »K« ist (Zeile 2320). Daher sehen Sie jetzt auch praktisch dieselbe Anordnung wieder. Die bereits herausgenommenen Elemente können dabei natürlich übersprungen werden (Zeile 2410).

Der Computer erreicht die Zeile 2440 nur, wenn Ihre Eingabe verwertet werden konnte. Und in diesem Fall springt er wieder an den Anfang, wo die Liste ausgegeben wird usw. (siehe oben). Die Zeilen 2450 bis 2470 werden nur abgearbeitet, wenn die Bedingungen in den Zeilen 2130, 2150, 2160 und 2170 erfüllt sind. Das heißt, das sind eher die negativen Fälle. Daher könnte es etwas verwirrend sein, daß die NEXT-Befehle am Ende stehen, als ob sie in jedem Fall ausgeführt würden. Wenn Ihnen das besser gefällt, können Sie den Befehl »GOTO 2450« durch die Befehlsfolge »NEXT J:NEXT I:RETURN« bzw. das »GOTO 2460« durch »NEXT I:RETURN« ersetzen. In dem Fall können Sie die Zeilen 2450 bis 2470 ersatzlos streichen sowie alle Einrückungen mit den Doppelpunkten und »SPACEs« entfernen. Dadurch könnte die Programmstruktur etwas deutlicher heraustreten.

Die Zeilen ab 2480 sind lediglich Fehlermeldungen für den Fall, daß Sie keine Zahlen mehr nehmen können. Das Programmende verdient auch noch eine Bemerkung. Sehen Sie sich dazu bitte die Zeilen 3050 bis 3080 an: In diesen Zeilen wird der neue Punktestand des Computers ausgerechnet. Dazu müssen seinem bisherigen Punktekonto noch alle übriggebliebenen Zahlen gutgeschrieben werden. Einfacher ist es aber, alle Punkte zu addieren, und davon Ihren Anteil zu subtrahieren. Und genau so arbeitet das Programm!

13 Drachensuche

Einführung

Haben Sie schon einmal etwas von Adventures gehört? Dabei handelt es sich um eine ganz spezielle Art von Spielen. Es geht meistens um eine kleine Geschichte, in deren Mittelpunkt ein bestimmter Held steht. Diese Geschichte wurde auf einen Computer umgesetzt, und somit sind Sie der Held der Geschichte. Allerdings ballern Sie sich nicht mit einem Joystick durch die Abenteuer, sondern Sie müssen versuchen, eine gestellte Aufgabe mit Geschick und viel Denken zu lösen. Dafür stellen Sie dem Computer Fragen oder erteilen ihm Befehle, die er dann für Sie ausführt.

Um ein solches Adventure handelt es sich bei Drachensuche. Allerdings ist es nicht besonders weit ausgeführt – das ist Ihre Aufgabe. Mit etwas Fantasie (und dem bereits vorhandenen Basic-Wissen) sollten Sie in der Lage sein, dieses Programm zu erweitern ...

Zuerst möchte ich Ihnen kurz die Vorgeschichte dieses Adventures erzählen: Sie leben ganz allein in der Wildnis und sind auf der Suche nach etwas Eßbarem. Dabei finden Sie die Spuren eines großen Drachens, der Sie, bei erfolgreicher Jagd, mindestens vier Wochen mit Fleisch versorgt. Darum folgen Sie den Spuren, auch wenn sie in ein unterirdisches Höhlensystem führen. Das Wort »Angst« fehlt in Ihrem Wortschatz. Sie dringen immer tiefer in die Höhle ein, und da sind Sie jetzt!

Die vorliegende Fassung ist ziemlich einfach gehalten. Der Computer kann zwischen 20 Sätzen wählen, die er Ihnen ausgibt. Anschließend möchte er wissen, ob Sie marschieren oder schießen wollen. In beiden Fällen müssen Sie danach noch eine Richtung (also »N«, »S«, »W« oder »O«) angeben. Anschließend wertet der Computer Ihren Zug aus; das heißt, er prüft, ob Sie den Ausgang des Höhlensystems gefunden haben, ob Sie den Drachen erlegt haben, ob ein Erdbeben stattfindet etc.

Danach sucht er sich einen weiteren Satz und fängt wieder von vorne an. Ziel des Spieles ist es, den Drachen zu erlegen und mit der Beute den Ausgang wiederzufinden. Allerdings lauern da immense Gefahren auf Sie; es soll Fledermäuse geben, die Sie durch die Höhlen verschleppen können, es soll plötzlich auftretende Abgründe geben...

Viel Erfolg!

Listing zu Drachensuche

```
10 REM *******************
20 REM ***             ***
30 REM *** DRACHENSUCHE ***
40 REM ***             ***
50 REM *******************
60 REM
70 GOSUB 1000
80 GOSUB 2000
90 GOSUB 3000
100 END
1000 REM
1010 REM *** VORBEREITUNGEN ***
1020 REM
1030 PRINT "{CLR,2DOWN}";TAB(13);"DRACHENSUCHE"
1040 DIM SA$(20), TR(20,4)
1050 FOR I=1 TO 20
1060 :READ A$,B$
1070 :SA$(I)=A$+B$
1080 NEXT I
1085 GOTO 1500
1090 DATA "SIE BEFINDEN SICH IN EINER NIEDRIGEN{4SPACE}HOEHLE. SIE IST V
     OLL VON
1100 DATA " FELSEN UND{5SPACE}GESTEINSTRUEMMERN."
1110 DATA "SIE MUESSEN TIEF GEBUECKT WEITERGEHEN - FAST SCHON KRIECHEN.
     SCHUTT"
1120 DATA " UND MUELL{3SPACE}VERSPERREN IHNEN DEN WEG."
1130 DATA "DER BODEN DER HOEHLE IST GLITSCHIG UND{2SPACE}STEIL. SIE MUES
     SEN AUF"
1140 DATA " ALLEN VIEREN{5SPACE}KRIECHEN."
1150 DATA "SIE MUESSEN SICH ZWISCHEN RIESIGEN{6SPACE}STALAGNITEN DURCHZW
     AENGEN;"
1160 DATA " IHRE KLEIDUNGHAENGT IHNEN SCHON IN FETZEN VOM LEIB!"
1170 DATA "SIE ERREICHEN EINEN RUNDEN SAAL MIT{5SPACE}MEHREREN"
1180 DATA " AUSGAENGEN."
1190 DATA "NEBEL ZIEHT AUF. PLOETZLICH VERLISCHT{3SPACE}"
1200 DATA "IHR LICHT - GEFAHR !!!!!!"
1210 DATA "EIN SCHWERES ERDBEBEN MUSS DIE HOEHLE{3SPACE}MITGENOMMEN HABEN, DA "
1220 DATA "GESTEINSTRUEMMER{3SPACE}AUF DEM BODEN LIEGEN."
1230 DATA "SIE KOMMEN AN EINEN WASSERGRABEN UND{4SPACE}MUESSEN DURCHSCHW
     IMMEN -"
1240 DATA " SIE ERREICHEN{2SPACE}DAS ANDERE UFER NUR MIT LETZTER MUEHE !"
1250 DATA "SIE KOMMEN IN EINEN HOHEN SAAL - IN DER MITTE ERKENNEN SIE EINEN"
1260 DATA " RIESIGEN FELSEN"
1270 DATA "EIN LOCH IN DER DECKE LAESST EINEN{6SPACE}FAHLEN LICHTSCHEIN"
1280 DATA " DURCHFALLEN."
1290 DATA "SIE SEHEN EIN LOCH IN DER WAND - LICHT{2SPACE}FAELLT{2SPACE}H
     INDURCH."
1300 DATA " SIE KOMMEN ABER NICHT HINDURCH; DAS LOCH IST EINFACH ZU ENG."
1310 DATA "IHR MAGEN KNURRT - BEIM NAECHSTEN{7SPACE}WASSERTRINKEN FALLEN SIE"
1320 DATA " VOR{12SPACE}ERSCHOEPFUNG IN DEN UNTERIRDISCHEN{6SPACE}WASSER
     LAUF."
1330 DATA "AUF EINMAL FINDEN SIE EINE ANGEZUENDETE KERZE - AB SOFORT"
1340 DATA " WIRD IHR WEG WIEDER{3SPACE}HELL ERLEUCHTET!"
1350 DATA "EIN RINNSAL SICKERT AUS EINER RITZE IN{2SPACE}DER"
1360 DATA " FELWAND."
1370 DATA "SIE SIND IN EINEM GEWUNDENEN STOLLEN.{3SPACE}DER BODEN IST"
1380 DATA " SCHLUEPFRIG UND MIT RISSEN UEBERZOGEN."
1390 DATA "DER GANG IST SEHR SCHMAL UND NIEDRIG.{3SPACE}SIE MUESSEN"
1400 DATA " SICH DURCHZWAENGEN."
1410 DATA "GIFTIGE DAEMPFE BEDROHEN SIE. SIE LAUFENGEFAHR,"
1420 DATA " OHNMAECHTIG ZU WERDEN."
1430 DATA "DIE DECKE DES STOLLENS SENKT SICH"
```

```
1440 DATA " IMMER TIEFER - SIE MUESSEN DARUNTER{11SPACE}DURCHKRIECHEN."
1450 DATA "SIE SIND IN EINEM STEIL ABFALLENDEN{5SPACE}STOLLEN."
1460 DATA " DER STOLLEN WIRD IMMER ENGER."
1470 DATA "SIE RIECHEN DEN MUNDGERUCH DES DRACHENS -"
1480 DATA " ES STINKT ENTSETZLICH !!!!"
1500 DEF FN R(X)=INT (RND(1)*X)+1
1510 FOR I=1 TO 20
1515 :F=0
1520 :FOR J=1 TO 4
1530 : IF FN R(3)=2 OR TR(I,J) THEN GOTO 1580
1540 : SA=FN R(20):IF SA=I THEN GOTO 1530
1550 : RI=FN R(4):IF TR(SA,RI) THEN GOTO 1530
1560 : TR (I,J)=SA
1570 : TR (SA,RI)=I
1580 : F=F+TR(I,J)
1590 :NEXT J
1600 :IF F=0  THEN GOTO 1520
1610 NEXT I
1620 SI=FN R(20)
1630 UG=FN R(20):IF UE THEN UG=-1
1640 EX=FN R(20)
1650 B1=FN R(20)
1660 B2=FN R(20)
1670 P1=FN R(20)
1680 P2=FN R(20)
1690 RETURN
2000 REM
2010 REM *** SPIEL ***
2020 REM
2030 PRINT:PRINT SA$(SI)
2040 FOR I=1 TO 4
2050 :CO=TR (SI,I)
2060 :IF CO=EX THEN PRINT "DER AUSGANG IST GANZ NAHE ..."
2070 :IF CO=UG THEN PRINT "ICH RIECHE DAS UNGEHEUER !!!"
2080 :IF CO=B1 OR CO=B2 THEN PRINT "SCHLUERF ... SCHLUERF ... SCHLUERF"
2090 :IF CO=P1 OR CO=P2 THEN PRINT "ICH SPUERE EINEN LUFTZUG !!!"
2100 NEXT I
2110 IF FN R(15)=4 THEN PRINT "*** ERDBE.B..EN ***":GOSUB 1620:GOTO 2000
2120 PRINT:AN$=""
2130 INPUT "M)ARSCHIEREN ODER S)CHIESSEN ";AN$
2140 IF AN$="M" THEN GOTO 2200
2150 IF AN$="S" THEN GOTO 2400
2160 PRINT "BITTE NUR M ODER S EINGEBEN !":GOTO 2130
2200 AN$=""
2210 INPUT "WELCHE RICHTUNG ";AN$
2230 GOSUB 2900:IF I=0 THEN GOTO 2200
2240 IF TR(SI,I)=0 THEN PRINT "DER WEG IST VERSPERRT ...":GOTO 2000
2250 PRINT "OK..."
2260 SI=TR(SI,I)
2270 IF SI=EX THEN WL=0:RETURN
2280 IF SI=UG THEN WL=1:RETURN
2290 IF SI=P1 OR SI=P2 THEN WL=2:RETURN
2300 IF SI<>B1 AND SI<>B2 THEN GOTO 2000
2310 PRINT "DIE FLEDERMAEUSE GREIFEN AN !!!"
2320 PRINT "SIE WERDEN HOCHGEHOBEN !"
2330 PRINT "...WO SIND SIE GELANDET ??"
2340 SI=FN R(20):GOTO 2000
2400 AN$=""
2410 INPUT "IN WELCHE RICHTUNG ";AN$
2420 GOSUB 2900:IF I=0 THEN GOTO 2400
2430 IF TR(SI,I) THEN GOTO 2460
2440 PRINT "KLONG !!"
2450 PRINT "DER PFEIL IST VON DER WAND ABGEPRALLT.":GOTO 2000
2460 IF TR(SI,I)<>UG THEN GOTO 2510
2470 PRINT "AHHH !"
2480 PRINT "SIE HABEN EIN UNGEHEUER ERLEGT !"
```

```
2490 PRINT "FINDEN SIE DEN AUSGANG ..."
2500 UE=1:UG=-1:GOTO 2000
2510 PRINT "IHR PFEIL HAT DAS UNGEHEUR VERFEHLT !":GOTO 2030
2900 AN$=LEFT$(AN$,1):I=0
2910 IF AN$="N" THEN I=1:RETURN
2920 IF AN$="O" THEN I=2:RETURN
2930 IF AN$="S" THEN I=3:RETURN
2940 IF AN$="W" THEN I=4:RETURN
2950 PRINT "BITTE NUR N,S,W ODER O EINGEBEN !"
2960 RETURN
3000 REM
3010 REM *** ENDE ***
3020 REM
3030 IF WL<>0 OR UE<>1 THEN GOTO 3060
3040 PRINT "SIE HABEN DEN AUSGANG MIT IHREM UNGE-"
3050 PRINT "HEUER ERREICHT. SIE WERDEN HEUTE EINE{3SPACE}GUTE MAHLZEIT G
     ENIESSEN !"
3055 RETURN
3060 IF WL<>0 THEN GOTO 3090
3070 PRINT "SIE HABEN DEN AUSGANG ERREICHT - LEIDER"
3080 PRINT "OHNE{2SPACE}UNGEHEUER. SIE MUESSEN HEUTE WOHL HUNGERN.":RETURN
3090 IF WL=2 THEN GOTO 3120
3100 PRINT "SIE SIND DEM UNGEHEUR DIREKT{2SPACE}IN DIE"
3110 PRINT "ARME GELAUFEN. ES HAT SIE GEFRESSEN.":RETURN
3120 PRINT "SIE SIND IN EINEN TIEFEN ABGRUND"
3130 PRINT "GESTUERZT - UND WURDEN NIE MEHR GESEHEN.":RETURN
```

■ Programmbeschreibung für Drachensuche

Wie Sie bereits wissen, handelt es sich bei diesem Programm um ein Adventure. Daher ist der Programmtext ziemlich undurchsichtig, da viel mit Variablen, Berechnungen und Ähnlichem gearbeitet wird. Ein Programm, das nur aus PRINT-Anweisungen besteht, ist ziemlich einfach zu verstehen. Aber sobald es nur noch aus Variablen und Algorithmen besteht, wird es sehr schwer, so ein Programm als »Außenstehender« nachzuvollziehen.

Dieses Programm ist selbstverständlich auch nicht so umfangreich wie ein professionelles Adventure, bei weitem auch nicht so ausgearbeitet. Aber ich hoffe, Sie haben bei den Spielen bis jetzt gemerkt, daß es nicht so sehr auf den Spielreiz ankommt. Natürlich freut es mich, wenn Sie ein Spiel manchmal noch herausholen und es als Zeitvertreib einmal spielen. Die Betonung liegt bei diesem Buch aber auf dem Lernerfolg. Und nur daher kann ich es wagen, ein solches »Mini-Adventure« in das Buch aufzunehmen.

Kommen wir aber endlich zum Spiel selbst. Nachdem der Titel ausgegeben worden ist, werden zwei Variablenfelder definiert, das eine als String-Array, das andere als ganz »normales« Zahlenfeld. In der anschließenden Schleife (Zeilen 1050 bis 1070) werden jeweils zwei Strings aus den DATA-Zeilen gelesen. Diese beiden Strings werden zu einem neuen verbunden und dem Feld »SA$(I)« zugewiesen. Dabei sehen Sie auch, wie einfach man mit Strings arbeiten kann. Das gilt natürlich nur für ein paar ausgesuchte Funktionen und Verknüpfungen. Beispielsweise können Sie kein Minuszeichen zwischen zwei Strings setzen, das führt zu einem »`?TYPE MISMATCH ERROR`«.

Die Variable »SA$« wird deshalb so »umständlich« belegt, damit man Sätze bilden kann, die länger sind als die Zeilenbegrenzung. Und zwar kann ein String bis zu 255 Zeichen aufnehmen. Eine Basic-Zeile kann aber (inklusive Zeilennummer) nicht mehr als 80 Zeichen lang sein. Wenn Sie jetzt einen String auf zwei Zeilen aufteilen und danach mittels »+« wieder zusammenfügen, können Sie entsprechend längere Sätze bilden.

Zu den DATA-Zeilen möchte ich nur eine kurze Bemerkung machen. Sie werden sich vielleicht etwas über die komische Verteilung der »SPACEs« wundern. An manchen Stellen stehen mehrere »SPACEs«, damit einzelne Wörter nicht am rechten Rand beginnen und am linken Rand in der nächst tieferen Zeile fortgesetzt werden.

Ich habe Ihnen bei Word Scramble bereits erklärt, daß ein Computer die DATA-Zeilen nicht unbedingt durchlaufen muß, um die Daten richtig einlesen zu können. Um das Programm ein klein wenig schneller zu machen, habe ich hinter den NEXT I-Befehl in Zeile 1080 ein »GOTO 1500« einfügt, damit die ganzen DATA-Zeilen übersprungen werden. Ab Zeile 1500 beginnt bereits der theoretische Teil. Hier wird der Grundstein für die Höhlen gelegt. Und zwar wird festgelegt, bei welchem Saal welche Meldung ausgegeben werden soll, in welche Richtungen man gehen kann usw.

Die Zufallsfunktion in Zeile 1500 ist noch nicht so schwer. Aber der Rest! Ich glaube auch, es bringt im Moment wenig, wenn ich versuche, Ihnen zu erklären, wie das Programm hier genau arbeitet. Die einen, die sich nicht für Adventures erwärmen können, langweilen sich zu Tode; und denjenigen, die sich für die Adventureprogrammierung brennend interessieren, rate ich, sich ein Spezialbuch zu kaufen. Mittlerweile gibt es zu diesem Thema (gerade für den C64) reihenweise Bücher und Sonderausgaben renommierter C64-Zeitschriften.

Aber in den Zeilen 1620 bis 1680 werden verschiedene Daten festgelegt, die ich Ihnen näher erläutern möchte. In der Variablen »SI« wird Ihr Standort zufällig ausgewählt (»SI« für Sie). In »UG« wird natürlich der Standort des Drachens festgelegt (»UG« wie UnGeheuer); Sie werden später noch sehen, daß das Programm unter Umständen noch einmal an diese Stelle zurückkehrt. Zu dem Zeitpunkt kann es dann schon sein, daß Sie das Ungeheuer erlegt haben. Ist das der Fall (»UE« für Ungeheuer Erlegt ungleich null), dann muß »UG« auf –1 gesetzt werden. »EX« (wie EXit, zu deutsch Ausgang) dürfte wohl klar sein. Die nächsten Variablen gehören immer paarweise zusammen. »B1« und »B2« sind für Fledermäuse zuständig, während »P1« und »P2« einen tiefen Abgrund signalisieren.

In Zeile 2030 wird erst einmal der Satz ausgegeben, der für Ihre derzeitige Position vorgesehen ist. Mit der anschließenden Schleife werden alle Möglichkeiten, die die Variable TR(SI,I) offenhält, durchlaufen. Es könnte sein, daß man dem Ausgang ganz nahe ist, daß man dem Ungeheuer ganz nahe ist, daß man den Fledermäusen ganz nahe ist oder daß man einem Abgrund ganz nahe ist (siehe auch Zeilen 2060 bis 2090). Die Variable »CO« wird ganz einfach eingeführt, um Tipparbeit zu sparen. Die Meldungen, welchen Gefahren Sie sich nähern, sind nicht unbedingt eindeutig. So zum Beispiel die Meldung, daß man sich einem Abgrund nähere. Unbedarfte Spieler können da leicht in Versuchung geführt werden, unter »ICH SPUERE EINEN LUFTZUG« den Ausgang zu verstehen.

In Zeile 2110 sehen Sie, wann ein Erdbeben die Höhle unsicher macht, nämlich in einem von 15 Fällen; wären alle Fälle gleichwahrscheinlich (was man bei dem Zufallsgenerator im C64 wahrlich nicht sagen kann!), könnte man behaupten, die Wahrscheinlichkeit für ein Erdbeben sei ein 15tel (na, das war ein Satz, ha?).

Schauen Sie sich aber die Zeile 2110 noch genauer an, da sehen Sie erst einmal, was ein solches Erdbeben alles anstellen kann. Es springt nämlich noch einmal in die Vorbereitungen, um einige Variablen neu zu belegen. Alle Plätze (für Sie, für das Ungeheuer ...) werden umgekrempelt.

In Zeile 2120 wird die Variabel »AN$« (ANtwort) zurückgesetzt. Wenn Sie wollen, daß Sie bei der Frage in 2130 nur [RETURN] drücken müssen, um beispielsweise weiterzumarschieren, dann können Sie hier die Variable entsprechend anders vorbelegen (mit »M« oder »S«). Nach der Frage wird Ihre Antwort ausgewertet. Alle anderen Buchstaben außer »M« und »S« werden aussortiert. Bei beiden Buchstaben verzweigt das Programm zu den entsprechenden Unterroutinen.

Ab Zeile 2200 steht die Routine für »Marschieren«. Hier können Sie auch das »AN$« wieder mit einer von Ihnen bevorzugten Marschrichtung vorbelegen, falls Sie Lust dazu haben. Diesmal wird Ihre Eingabe aber nicht direkt ausgewertet, sondern noch ein anderes Unterprogramm verwendet; bei dem Programmteil »Schießen« muß ja auch noch einmal nach der Schußrichtung gefragt werden. Darum ist es einfacher, diese Auswertung getrennt zu programmieren und dann von beiden Teilen aufrufen zu lassen.

In der Variablen »I« wird das Ergebnis der Auswertung übergeben. Sollte diese Variable null sein, war irgendetwas mit der Eingabe nicht in Ordnung. Ihre Eingabe wird – soweit in Ordnung – mit den Örtlichkeiten verglichen. Es könnte ja sein, daß der Weg versperrt ist (siehe Zeile 2240).

Normalerweise können Sie aber in jede beliebige Richtung gehen. Und die Auswertung Ihrer Wanderung erfolgt in den Zeilen 2270 bis 2300. Auch hier wird der Variablen SI der Inhalt von TR(SI,I) zugewiesen. Zum einen, weil das sowieso der neue Inhalt von »SI« ist, und zum anderen, um Tipparbeit zu sparen.

Während in den Abfragen der Zeilen 2060 bis 2090 die Gefahren erst in der Nähe waren, sind sie jetzt ganz akut. Daher wird auch gleich zurückgekehrt, sollte eine der Bedingungen erfüllt sein. Zuvor wird jedoch noch eine Variable »WL« gesetzt, mit der man beim Programmende ab Zeile 3000 noch differenzierte Kommentare ausgeben kann.

Es kann natürlich sein, daß nichts Großartiges passiert. In dem Fall springt das Programm in Zeile 2300 einfach wieder nach Zeile 2000. Sollten Sie von den Fledermäusen angegriffen werden, muß Ihre neue Position erst wieder bestimmt werden (siehe Zeile 2340). Alle anderen Daten (vom Drachen, vom Ausgang etc.) bleiben natürlich erhalten (Fledermäuse sind noch etwas anderes als Erdbeben).

Die Routine für den Schuß ist so ähnlich aufgebaut wie die für das Marschieren. Allerdings muß hier natürlich unterschieden werden, ob Sie das Ungeheuer treffen oder nicht. Die Ab-

fragen überlasse ich diesmal Ihnen. Wie weiter oben schon angesprochen, wird in der Variablen »UE« gespeichert, ob das Ungeheuer bereits erlegt wurde. Wenn »UE« auf 1 gesetzt wird, muß »UG« den Wert -1 annehmen.

Am Ende des Programms sollen noch differenzierte Meldungen ausgegeben werden, je nach Spielverlauf. Dazu empfiehlt sich eine kleine Übersicht:

WL	= 0	Ausgang gefunden
WL	= 1	Vom Ungeheuer gefressen
WL	= 2	In Abgrund gestürzt
UE	= 0	Kein Ungeheuer erlegt
UE	= 1	Fette Beute!

Und je nachdem, wie diese Variablen besetzt sind, wird entschieden, welche Meldung ausgegeben werden soll. Strengen Sie sich ein bißchen an!

14 Drachenjagd

Einführung

Wie es der Titel schon andeutet, ist dieses Programm dem vorausgehenden recht ähnlich. Dennoch gibt es grundlegende Unterschiede. Man könnte vielleicht sagen, daß die »Hintergrundgeschichten« der beiden Programme ähnlich sind.

Bei dieser Version erhalten Sie auf alle Fälle keine Textausgabe. Vielmehr befinden Sie sich in einem unsichtbaren Labyrinth. Darin gibt es Felder, die Sie an einen völlig unberechenbaren Ort versetzen, es gibt Dynamitfelder... Ansonsten gibt es nur noch den Ausgang, den Drachen Grusi und Sie.

Sie haben die Wahl zwischen weitergehen oder auf den Drachen schießen. In beiden Fällen werden Sie nach der Richtung gefragt. Beachten Sie bitte, für rückwärts nicht »R«, sondern »U« einzugeben; »R« wurde bereits für »rechts« benötigt. Anschließend darf Grusi ziehen. Sein Ziel ist es, Sie zu erwischen. Daher bewegt er sich auch nicht zufallsgesteuert, sondern zielstrebig auf Sie zu. Vielleicht sollten Sie sich beeilen, den Ausgang so schnell wie möglich zu finden.

Als Hilfe wird immer der Abstand zwischen Ihnen und dem Ausgang sowie dem Drachen Grusi angezeigt. Aus den Veränderungen nach einem Zug können Sie schließen, in welcher Richtung von Ihnen aus der Ausgang bzw. der Drache liegt (oder steht, je nachdem).

Sehr viel mehr läßt sich zu dem Programm nicht sagen ... außer, daß Sie fast keine Chance haben!

Listing zu Drachenjagd

```
10 REM ******************
20 REM ***            ***
30 REM *** DRACHENJAGD ***
40 REM ***            ***
50 REM ******************
60 REM
70 GOSUB 1000
80 GOSUB 2000
90 END
1000 REM
1010 REM *** VORBEREITUNGEN ***
1020 REM
1030 PRINT"{CLR,2DOWN}";TAB(14);"DRACHENJAGD"
1040 DIM MA(15,15)
1050 DEF FN R(X)=INT (RND(1)*X)+1
1060 DEF FN A(X)=0.001* INT(X*1000+0.5)
1100 REM ********** BETRETEN UNMOEGLICH
1110 FOR I=1 TO 20
1120 :X=FN R(15):Y=FN R(15)
1130 :IF MA(X,Y)<>0 THEN GOTO 1120
1140 :MA(X,Y)=1
1150 NEXT I
1200 REM ******************* VERSETZUNG
1210 FOR I=1 TO 20
1220 :X=FN R(15):Y=FN R(15)
1230 :IF MA(X,Y)<>0 THEN GOTO 1220
1240 :MA(X,Y)=2
1250 NEXT I
1300 REM ********************** DYNAMIT
1310 X=FN R(15):Y=FN R(15)
1320 IF MA(X,Y)<>0 THEN GOTO 1310
1330 MA(X,Y)=3
1400 REM ********************** AUSGANG
1410 XN=FN R(15):YN=FN R(15)
1420 IF MA(XN,YN)<>0 THEN GOTO 1410
1430 MA(XN,YN)=4
1500 REM *********************** GRUSLI
1510 XG=FN R(15):YG=FN R(15)
1520 IF MA(XG,YG)<>0 THEN GOTO 1510
1530 MA(XG,YG)=5
1600 REM ********************** SPIELER
1610 XS=FN R(15):YS=FN R(15)
1620 IF MA(XS,YS)<>0 THEN GOTO 1610
1630 MA(XS,YS)=5
1640 SP=0:GR=0
1650 RETURN
2000 REM
2010 REM *** SPIEL ***
2020 REM
2030 GOSUB 3000
2040 GOSUB 3100
2050 IF WL THEN RETURN
2060 PRINT
2070 AN$="":INPUT "ZIEHEN ODER SCHIESSEN (Z/S) ";AN$
2080 IF AN$<>"Z" AND AN$<>"S" THEN GOTO 2070
2090 PRINT "VORWAERTS, RUECKWAERTS,"
2100 INPUT "RECHTS ODER LINKS (V/U/L/R) ";RI$
2110 IF RI$<>"V" AND RI$<>"U" AND RI$<>"R" AND RI$<>"L" THEN GOTO 2100
2120 IF RI$="V" THEN X=0:Y=-1:GOTO 2160
2130 IF RI$="U" THEN X=0:Y=1:GOTO 2160
2140 IF RI$="R" THEN X=1:Y=0:GOTO 2160
2150 IF RI$="L" THEN X=-1:Y=0
```

```
2160 IF AN$="S"THEN GOTO 2500
2170 X=X+XS:Y=Y+YS
2180 IF X>=1 AND X<=15 AND Y>=1 AND Y<=15 THEN GOTO 2210
2190 PRINT "DIESER ZUG LIEGT AUSSERHALB DES BEREICHS."
2200 GOTO 2800
2210 IF MA(X,Y)<>1 THEN GOTO 2240
2220 PRINT "DIESE STELLE KOENNEN SIE NICHT BETRETEN."
2230 GOTO 2800
2240 IF MA(X,Y)<>2 THEN GOTO 2280
2250 PRINT "SIE WERDEN IRGENDWOHIN VERSETZT ..."
2260 X=FN R(15):Y=FN R(15)
2265 IF MA(X,Y)=1 THEN GOTO 2260
2270 GOTO 2240
2280 IF MA(X,Y)<>3 THEN GOTO 2320
2290 PRINT "DER ZUG IST ZWAR ERLAUBT, ABER ER KOSTET";
2300 PRINT "SIE IHR LEBEN, TJA DYNAMIT !"
2310 RETURN
2320 IF MA(X,Y)<>4 THEN GOTO 2370
2330 PRINT "ZUG IST ERLAUBT."
2340 PRINT "SIE FANDEN DEN AUSGANG !"
2350 PRINT "SOMIT SIND SIE ENDLICH WIEDER FREI !!!"
2360 RETURN
2370 IF MA(X,Y)<>5 THEN GOTO 2410
2380 PRINT "ZUG IST ERLAUBT."
2390 GOSUB 3110
2400 RETURN
2410 PRINT "ZUG IST ERLAUBT."
2420 MA(XS,YS)=SP
2430 XS=X:YS=Y
2440 SP=MA(XS,YS):MA(XS,YS)=6
2450 GOTO 2800
2500 SX=XS:SY=YS
2510 SX=SX+X:SY=SY+Y
2520 PRINT "ZISCHHHHHHHHHH";
2530 IF SX>=1 AND SX<=15 AND SY>=1 AND SY<=15 THEN GOTO 2570
2540 PRINT " WUSCHHHHHHHH";
2550 PRINT " DER SCHUSS VERLIESS DAS LABYRINTH."
2560 GOTO 2800
2570 IF MA(SX,SY)=0 OR MA(SX,SY)=2 OR MA(SX,SY)=3 OR MA(SX,SY)=4 THEN GO
     TO 2510
2580 IF MA(SX,SY)<>1 THEN GOTO 2630
2590 PRINT "DUSCHHHHHHHHHH !"
2600 PRINT "DER PFEIL PRALLTE GEGEN EINE MAUER !"
2610 PRINT "GRUSI WURDE NICHT GETROFFEN."
2620 GOTO 2800
2630 PRINT "AHHHHHHH !"
2640 PRINT"SIE HABEN GRUSI VERLETZT UND VON DIESEM FELD GEJAGT."
2650 MA(SX,SY)=GR
2660 XG=FN R(15):YG=FN R(15)
2670 GR=MA(XG,YG):MA(XG,YG)=5
2800 REM *********** GRUSI ZIEHT SELBER
2810 GOSUB 3000
2820 PRINT "{DOWN}GRUSI ZIEHT JETZT ..."
2830 GOSUB 3100
2840 IF WL THEN RETURN
2850 IF XS<XG THEN X=-1:Y=0:GOTO 2890
2860 IF XS>XG THEN X=1:Y=0:GOTO 2890
2870 IF YS<YG THEN X=0:Y=-1:GOTO 2890
2880 IF YS>YG THEN X=0:Y=1
2890 MA(XG,YG)=GR
2900 XG=XG+X:YG=YG+Y
2910 GR=MA(XG,YG):MA(XG,YG)=5
2920 GOTO 2030
3000 AG=FN A(SQR((XG-XS)↑2 + (YG-YS)↑2))
```

```
3010 AA=FN A(SQR((XS-XN)↑2 + (YS-YN)↑2))
3020 PRINT "{DOWN}GRUSI IST";AG;"M WEIT WEG."
3030 PRINT "{DOWN}DER AUSGANG IST";AA;"M WEIT WEG."
3040 RETURN
3100 IF AG>=2 THEN RETURN
3110 PRINT "{DOWN,SPACE}<<< SCHLABBER >>>"
3120 PRINT "GRUSI WURDE SEINEM NAMEN GERECHT UND HAT";
3130 PRINT "SIE GANZ GRUSELIG AUFGESAUGT !"
3140 PRINT "SIE HABEN WOHL VERLOREN !"
3150 WL=1:RETURN
```

■ Programmbeschreibung für Drachenjagd

Dieses Spiel beginnt zunächst mit einer Variablendimensionierung. Das Array MA soll zweidimensional sein und jeweils 15 Elemente enthalten (Zeile 1040). Daraufhin werden zwei Funktionen definiert; die erste, »R(X)«, stellt mal wieder eine Zufallsfunktion dar. Welche Zahlen können Sie dabei erhalten?

Die andere Funktion A(X) rundet den Zahlenwert, den sie erhält. In den Zeilen 1100 bis 1640 werden die einzelnen Elemente des Labyrinths »konstruiert«. Zunächst sollen 20 Mauern verteilt werden, die man nicht »betreten« kann (es handelt sich wohl eher um Blöcke, die einem das Betreten der Felder unmöglich machen). Danach sollen noch Gummimauern verteilt werden, die einen irgendwohin ins Labyrinth verschlagen, wenn man gegen sie läuft. An einem Beispiel möchte ich Ihnen die Funktionsweise dieser Teile erläutern; somit kann ich mir die restlichen Teile sparen.

Den Rahmen bildet eine Schleife, die 20 mal durchlaufen wird. So viele Mauern sollen also verteilt werden. Bei jedem Durchlauf wird den Variablen »X« und »Y« jeweils ein Zufallswert übergeben. Ist dieses Feld noch nicht belegt (MA(X,Y)=0), dann kann darauf die Mauer errichtet werden (Zeile 1140). Andernfalls müssen zwei neue Zufallswerte berechnet werden (Zeile 1130).

Nach diesem Muster funktionieren alle weiteren »Bauarbeiten«. Als nächstes werden 20 Gummimauern, eine Stange Dynamit, ein Ausgang, der Drache Grusi und Sie im Labyrinth plaziert. Ich hoffe, Sie haben dabei festgestellt, daß für den Ausgang, Grusi und »Sie« eigene Variablen eingeführt werden, die die entsprechenden Koordinaten das ganze Spiel über beinhalten! Auf die Variablen »SP« (wie SPieler) und »GR« (wie GRusi) werde ich später noch genauer eingehen.

Im Programmteil »SPIEL« werden zunächst zwei Unterprogramme aufgerufen (Zeilen 2030 und 2040). Das erste berechnet Ihren Abstand sowohl vom Drachen als auch vom Ausgang. Die zweite Routine (ab Zeile 3100) prüft, ob Sie den Sicherheitsabstand zum Drachen einhalten und setzt dementsprechend eine Variable »WL«. Diese Variable »WL« wird in Zeile 2050 gleich abgefragt. Erinnern Sie sich noch, was diese »verkrüppelte« IF-Abfrage bedeutet? Sie ist erfüllt, sobald »WL« ungleich null ist. Das Programm setzt WL auf eins, wenn Sie dem Drachen zu nahe gekommen sind. Das bedeutet also, die Bedin-

gung in Zeile 2050 gilt als erfüllt, und das Programm wird beendet. Zeile 2060 gibt lediglich eine Leerzeile auf dem Bildschirm aus.

Anschließend wird gefragt, ob Sie ziehen oder auf den Drachen schießen wollen. Auch hier gilt das bei »Drachensuche« Gesagte: Wenn Sie bevorzugt weitergehen oder herumballern, können Sie den entsprechenden Wert in Zeile 2070 der Variablen »AN$« zuweisen, damit Sie bei der Frage nur noch [RETURN] zu drücken brauchen; das bedeutet allerdings nicht, daß Sie für die weniger beliebte Tätigkeit auch nur [RETURN] eingeben müssen. Sie können sich das Leben nur für eine Funktion leichter machen!

Sobald Sie diese erste Frage beantwortet haben, folgt sofort eine zweite, die wissen will, wohin Sie ziehen oder schießen wollen (RI$ wie RIchtung). In Zeile 2110 werden alle möglichen Tippfehler abgefangen, da nur die Tasten [V], [U], [L] und [R] durchgelassen werden.

In Zeile 2120 erfolgt dann die Auswertung Ihrer Eingabe. Dazu werden zwei neue Variablen eingeführt, das »X« und das »Y«. Beide geben zunächst an, wie sich Ihre Koordinaten ändern (zum Beispiel bei [V] bleibt Ihre X-Koordinate, während »Y« um eins erniedrigt wird).

Die GOTOs am Ende der ersten drei Abfragen sollen das Programm lediglich etwas beschleunigen. Sobald Ihre Eingabe beispielsweise als [V] erkannt wurde, müssen die übrigen drei Abfragen nicht auch noch durchlaufen werden. Daher kann man gleich zu den nächsten Befehlen springen, die wieder unabhängig von der Eingabe erfolgen. Sollten Sie bei der ersten Eingabe ein [S] für »Schießen« eingegeben haben, verzweigt Ihr Programm nach Zeile 2500. Beachten Sie bitte, daß dennoch die Richtungsangaben bereits ausgewertet wurden.

Sollten Sie zu diesem Zeitpunkt das Gehen bevorzugen, werden die Zeilen ab 2170 weiter bearbeitet. Dazu wird zu Ihren Koordinaten »XS« und »YS« (für X/Y-Koordinate des Spielers) jeweils die Veränderung »X« und »Y« addiert. Das Ergebnis dieser Addition wird wiederum »X« bzw. »Y« zugeordnet. Das dient dazu, daß man erst einmal überprüfen kann, wohin Sie mit Ihren Schritten überhaupt gelangen und diesen Schritt wieder rückgängig machen kann. Denn bis zur Überprüfung ist Ihr alter Standort immer noch in »XS« und »YS« gespeichert.

Wie gesagt beginnt jetzt die Überprüfung Ihres neuen Standortes. Zunächst wird getestet, ob dieser neue Platz innerhalb des Labyrinths liegt (Zeilen 2180 bis 2200).

Anschließend wird nachgeschaut, auf welcher Art von Feld Sie sich dann befinden werden. Wenn es sich dabei um einen Mauerblock handelt (Zeilen 2210 bis 2230), kann diese Stelle nicht betreten werden (vergleiche auch die Belegung des Labyrinths mit Werten (Zeilen 1100 bis 1630); gerade diese Belegung ist es, die im folgenden überprüft wird). Bei der Gummimauer (Zeilen 2240 bis 2270) werden Sie irgendwohin verschlagen. Dazu muß der C64 erst wieder neue Koordinaten für Sie auswählen. Er setzt Sie wahllos irgendwohin. Sollten Sie dabei auf eine Mauer stoßen (wohin Sie also gar nicht dürfen), müssen die neuen Koordinaten gleich noch einmal berechnet werden (Abfrage in 2265). Ansonsten

springt der C64 nach 2240, damit auch die neuen Koordinaten weiterhin berechnet werden. Ab hier wird also zwischen den Zufalls- und den aus Ihren Eingaben berechneten Koordinaten nicht mehr unterschieden.

Wenn Sie auf ein Feld mit Dynamit stoßen (Zeilen 2280 bis 2310), kehrt das Programm zum GOSUB-Befehl zurück, trifft dort als nächstes auf den END-Befehl und bricht demnach das Programm ab. Sollten Sie den Ausgang getroffen haben (Sie Glücklicher!), wird das Programm auch abgebrochen (Zeilen 2320 bis 2360).

Anschließend wird noch geprüft, ob Sie sich auf dem Feld des Grusli befinden werden (Zeile 2370). Sollte das der Fall sein, kommt die lapidare Meldung, daß der Zug erlaubt ist. Danach springt der Computer aber in die Routine ab Zeile 3110, das heißt, er benutzt lediglich die letzten paar Zeilen des Unterprogramms ab 3100. Und was dort steht, dürfte kein Anlaß zu übermäßiger Freude sein ...

Eine Anmerkung noch: Betrachten Sie bitte die Zeilen 2390 und 2400. Erinnern Sie sich? Ich habe Ihnen einmal gesagt, wenn ein RETURN-Befehl direkt auf ein »GOSUB« folgt, kann man das »RETURN« ersatzlos streichen und das »GOSUB« durch ein »GOTO« ersetzen. Wenn es beliebt?

Falls sich keine Einwände gegen Ihren Zug finden lassen, sollte der Interpreter mittlerweile bei Zeile 2410 angekommen sein. Auch dort wird Ihnen die oben bereits aufgeführte Meldung »ZUG IST ERLAUBT« auf dem Bildschirm ausgegeben. Anschließend kommt die Variable »SP« zum ersten Mal ins Spiel.

In »SP« wird gespeichert, welchen Wert das Feld hat, das Sie betreten. Wenn wir das Ganze einmal auflisten, um einen Überblick zu erhalten, sieht das, wie folgt, aus:

XS, YS	Ihre alten Koordinaten (vor diesem Zug)
X, Y	Neue Koordinaten (nach diesem Zug)
SP	Enthält den Wert, den das alte Feld ursprünglich hatte (MA(XS,YS))
MA(X,Y)	Ist das neue Feld, das Sie gleich betreten.

Dadurch sollten die folgenden paar Zeilen klar werden: Zunächst wird dem alten Feld MA(XS, YS) sein ursprünglicher Wert wieder zugewiesen. Schließlich soll alles so verlassen werden, wie es vorgefunden wurde.

Die Koordinaten »SX« und »SY« werden aktualisiert (Zeile 2430). Damit wurde Ihr Schritt endgültig besiegelt. »SP« wird der »ursprüngliche« Wert des neuen Feldes zugewiesen, damit es nach dem nächsten Schritt wieder ordentlich verlassen werden kann. Und das neue Feld wird noch mit dem »Ihnen eigenen Code« belegt. Zu guter Letzt erfolgt nur noch ein Sprung zu der Routine, die den guten Grusli weiter bewegt.

Ab Zeile 2500 stehen die Befehle für Ihren Schuß. Dazu werden den Variablen »SX« und »SY« (»S«, da wir in der »Schußabteilung« sind) Ihre derzeitigen Koordinaten zugeordnet (»S« für Spieler). In diesem Fall werden sich Ihre Koordinaten ja nicht verändern (da Sie sich nur mit Pfeil und Bogen gegen den Drachen verteidigen können, gibt es keinen so starken Rückstoß, der Sie auf ein anderes Feld werfen könnte).

Was sich allerdings verändert, sind die Schußkoordinaten. Da sie in einer bestimmten Richtung verändert werden, werden auch hier die beiden Variablen »X« und »Y« verwendet (siehe auch Zeilen 2120 bis 2150).

Bei jedem Schuß wird prinzipiell die Meldung »ZISCHHHHHHHHHH«; ausgegeben, damit Sie wissen, »was Sache ist«. Daraufhin wird überprüft, ob Ihr Pfeil sich noch innerhalb des Labyrinths befindet. Wenn nicht, erhalten Sie die Meldung » WUSCHHHHHHHH ...«. Der Computer springt in diesem Fall direkt nach Zeile 2800, um Grusi nicht einrosten zu lassen.

Sollte das Feld nichts Besonderes aufweisen, werden die Pfeilkoordinaten wieder entsprechend der Richtung verändert, und der Computer springt dafür zur Zeile 2510. Dabei stellen Gummiwände, Dynamit oder der Ausgang keine Hindernisse für Ihre Pfeile dar, von »ordinären« Feldern ohne spezielle Bedeutung wollen wir hier erst gar nicht reden.

Sollte keine dieser Bedingungen erfüllt sein, kann Ihr Pfeil nur noch gegen eine richtige Mauer geprallt sein oder Grusli erwischt haben. Diese beiden Fälle werden noch ab Zeile 2580 unterschieden. Sollten Sie bei diesen letzten Abfragen Probleme mit der Belegung des Arrays »MA« gehabt haben, so können Sie sich ruhig noch einmal die Vorbelegung in den Zeilen 1100 bis 1630 anschauen. Dort werden Sie das Nötige finden.

Sollten Sie Grusli erwischt haben, müssen noch ein paar Befehle abgearbeitet werden, da Sie Grusli nicht töten, sondern höchstens verjagen können. Zunächst wird auch diesem Feld sein ursprünglicher Wert wiedergegeben. Der wurde nicht in »SP«, sondern in »GR« gespeichert. Die neuen Koordinaten werden per Zufall gesteuert. Diesmal erfolgt keine Überprüfung, folglich ist es Grusli egal, ob er sich auf Dynamit oder eine Gummiwand oder gar auf Sie setzt (er walzt alles nieder). Schließlich muß in »GR« noch der Wert dieses Feldes gerettet werden, und diesem der Wert fünf (für Drachen) zugeordnet werden.

Jetzt kommen wir zu der Routine, die Grusli über das Spielfeld schiebt (ab Zeile 2800). Dazu wird zunächst ausgegeben, wie weit Sie vom Drachen bzw. Ausgang entfernt sind. Außerdem werden Sie mit einer Meldung auf der Mattscheibe darüber belehrt, daß Sie im Moment nicht an der Reihe sind. Bevor der Drache überhaupt zieht, testet das Programm, ob Sie ihm nicht vielleicht schon ziemlich nahe gekommen sind. Dazu verwendet es das Unterprogramm ab Zeile 3100. Sollte nach dem Unterprogramm »WL« gesetzt sein, muß abgebrochen werden (Zeile 2840).

Normalerweise kann Grusli jetzt aber in Aktion treten. Um seine Richtung zu bestimmen, vergleicht er seine Koordinaten mit den Ihren. Auch hier werden für die Veränderung die Variablen »X« und »Y« herangezogen.

Bevor sein Schritt jedoch ausgeführt wird, muß dem alten Feld der ursprüngliche Wert wieder zugewiesen werden (auch Drachen sind sorgfältig); dieser Ursprungswert ist noch in »GR« gespeichert! Erst danach können die Koordinaten auf den neuesten Stand gebracht werden (Zeile 2900). Damit auch später dieses Feld ohne Schäden verlassen werden kann, muß dessen Inhalt in »GR« wieder abgelegt werden. Dieses Feld wird jetzt vom Drachen gesetzt, daher erhält es den Wert fünf. Diese drei Zeilen 2890 entsprechen im Großen und

Ganzen den Zeilen 2650 bis 2670. Sie sind analog aufgebaut, verwenden höchstens ein paar unterschiedliche Variablen. Aber das macht auch deutlich, daß es dem Drachen nicht nur auf der Flucht vor Ihren Pfeilen, sondern prinzipiell egal ist, auf welchem Feld er sich niederläßt.

Das wäre Ihre Chance, das Programm etwas zu erweitern. Wenn Sie beispielsweise Drachenfallen im Labyrinth aufstellen könnten, die den Drachen entweder ganz »neutralisieren« oder für vier, fünf Runden außer Gefecht setzen...

Ein abschließendes »GOTO 2030« springt wieder an den Anfang, um Sie wieder zum Zuge kommen zu lassen. Zu guter Letzt kommen da noch die zwei Unterprogramme, die zum einen den Abstand Spieler-Drachen und Spieler-Ausgang berechnen bzw. ein grusliges Erlebnis zum besten geben.

Bei den Funktionen (Zeilen 3000 und 3010) wird wieder eine schon früher angesprochene geometrische Formel verwendet, deren Herleitung ich in diesem Rahmen unmöglich beschreiben kann. Ansonsten müßten Sie in der Lage sein, die restlichen paar Zeilen für sich zu analysieren!

15 Packman

Einführung

Ich glaube, jeder Computerbesitzer hatte schon einmal eine Packman-Version auf seinem Computer laufen. Daher hat dieses Programm keine Chance, wenn es um den Spielreiz geht, da es keinesfalls mit professionellen Spielen konkurrieren kann; das ist aber auch nicht Ziel dieses Buches.

Die Spielidee wurde bei dieser Version beibehalten: Es geht darum, mit einer kleinen Figur auf dem Bildschirm möglichst viele Punkte zu »fressen«, bevor die Geister kommen, um den Packman selbst aufzufressen. Allerdings kommt in diesem Spiel keine Figur vor, sondern der Packman wird dargestellt von einer Schlange, die um so länger wird, je mehr sie zu fressen bekommt.

Die Geister bewegen sich mehr oder weniger zielstrebig auf Sie zu – Sie sollten aber große Chancen haben, zwischen den Mauerblöcken »entschwinden zu können«, da diese Verfolgung meistens auf Zufall beruht.

Ihre Schlange steuern Sie übrigens mit:

[A] nach oben
[Z] nach unten
[J] nach links
[K] nach rechts

Ein kleiner Tip noch zur Steuerung: Es empfiehlt sich, nicht wie blöde auf die Tasten einzuhämmern. Vielmehr sollten Sie immer erst dann eine Taste drücken, wenn der Packman kurz vor einer Wegkreuzung steht. Nur an diesen Stellen (am Anfang sind sie noch durch Nahrungspunkte gekennzeichnet) können Sie die Richtung ändern. Dadurch soll ein Durchpflügen des »schönen« Bildschirmaufbaus verhindert werden.

Listing zu Packman

```
10 REM ***************
20 REM ***         ***
30 REM *** PACKMAN ***
40 REM ***         ***
50 REM ***************
60 REM
70 GOSUB 1000
80 GOSUB 2000
90 GOSUB 3000
100 END
1000 REM
1010 REM  *** VORBEREITUNGEN ***
1020 REM
1030 DIM CO(3,2), SP(3)
1040 DIM FA$(15)
1050 FOR I=0 TO 15
1060 :READ FA$(I)
1070 NEXT I
1080 DATA "{BLACK}","{WHITE}","{RED}","{CYAN}","{PURPLE}","{GREEN}","{BL
     UE}","{YELLOW}"
1090 DATA "{ORANGE}","{BROWN}","{LIG.RED}","{GREY 1}","{GREY 2}","{LIG.G
     REEN}","{LIG.BLUE}","{GREY 3}"
1100 PT=0
1110 PM=3:NM=4
1120 POKE 650,128
1130 POKE 53280,0:POKE 53281,0
1140 PRINT "{CLR,RVSON,WHITE,39SPACE}"
1150 FOR I=1 TO 19 STEP 3
1160 :POKE 211,0:POKE 214,I:SYS 58640
1170 :PRINT "{RVSON,WHITE,SPACE,RVOFF,37SPACE,RVSON,SPACE}"
1180 :PRINT "{RVSON,WHITE,SPACE,RIGHT,LIG.BLUE,3SPACE,RIGHT,3SPACE,RIGHT
     ,3SPACE,RIGHT,3SPACE,RIGHT,3SPACE,RIGHT}";
1190 :PRINT "{3SPACE,RIGHT,3SPACE,RIGHT,3SPACE,RIGHT,3SPACE,RIGHT,WHITE,
     SPACE}"
1200 :PRINT "{RVSON,WHITE,SPACE,RIGHT,LIG.BLUE,3SPACE,RIGHT,3SPACE,RIGHT
     ,3SPACE,RIGHT,3SPACE,RIGHT,3SPACE,RIGHT}";
1210 :PRINT "{3SPACE,RIGHT,3SPACE,RIGHT,3SPACE,RIGHT,3SPACE,RIGHT,WHITE,
     SPACE}"
1220 NEXT I
1230 PRINT "{RVSON,WHITE,SPACE,RVOFF,37SPACE,RVSON,SPACE}"
1250 PRINT "{GREEN,SPACE}Z Z Z";
1260 DIM PM (8,2)
1270 FOR I=5 TO 33 STEP 4
1280 :FOR J=4 TO 19 STEP 3
1290 : F$=MID$("{LIG.RED,LIG.GREEN,YELLOW,PURPLE,GREEN}",RND(1)*5+1,1)
1300 : POKE 211,I:POKE 214,J:SYS 58640
1310 : PRINT "{RVOFF}";F$;"*"
1320 :NEXT J
1330 NEXT I
1340 DEF FN BS(V)=PEEK(1024+X2+40*Y2+V)
1350 DEF FN FA(V)=PEEK(55296+X2+40*Y2+V) AND 15
1360 RETURN
2000 REM
2010 REM *** SPIEL ***
2020 REM
2030 CO(1,1)=1:CO(1,2)=1
2040 CO(2,1)=37:CO(2,2)=1
2050 CO(3,1)=37:CO(3,2)=22
2060 DE (1)=1:DE (2)=4:DE (3)=3
2070 FOR I=1 TO 3
2080 :POKE 211,CO(I,1):POKE 214,CO(I,2):SYS 58640
2090 :PRINT "{WHITE}W";
```

```
2100 :SP$(I)=" "
2110 NEXT I
2120 FOR I=1 TO NM
2130 :PM(I,1)=I
2140 :PM(I,2)=22
2150 NEXT I
2160 MX=NM:MY=22
2170 FOR I=1 TO NM
2180 :POKE 211,PM(I,1):POKE 214,PM(I,2):SYS 58640
2190 :PRINT "{GREEN}Z̲";
2200 NEXT I
2210 PR=1:KL=0:MM=0
2220 PM=PM-1:IF PM<0 THEN RETURN
2230 POKE 211,15:POKE 214,24:SYS 58640
2240 PRINT "{WHITE}<<< FERTIG >>>";
2250 FOR I=1 TO 2000:NEXT I
2260 PRINT "{15LEFT,15SPACE}";
2270 GOSUB 2480
2280 IF PT=48 THEN RETURN
2290 GOSUB 2700
2300 IF KL=0 THEN GOTO 2270
2310 FOR I=1 TO 3
2320 :POKE 211,CO(I,1):POKE 214,CO(I,2):SYS 58640
2330 :PRINT SP$(I);
2340 NEXT I
2350 FOR I=1 TO NM
2360 :POKE 211,PM(I,1):POKE 214,PM(I,2):SYS 58640
2370 :PRINT " ";
2380 NEXT I
2390 POKE 211,2*PM+1:POKE 214,23:SYS 58640
2400 PRINT "{2SPACE}";
2410 GOTO 2030
2480 GT=((MX+3)/4)=INT((MX+3)/4) AND ((MY+2)/3)=INT((MY+2)/3)
2490 IF NOT GT THEN GOTO 2510
2500 GET K$:IF K$="" THEN GOTO 2510
2501 IF K$="K" THEN PR=1
2502 IF K$="A" THEN PR=2
2503 IF K$="J" THEN PR=3
2504 IF K$="Z" THEN PR=4
2510 X2=MX:Y2=MY
2520 IF PR=1 THEN X2=X2+1
2530 IF PR=2 THEN Y2=Y2-1
2540 IF PR=3 THEN X2=X2-1
2550 IF PR=4 THEN Y2=Y2+1
2560 IF X2<1 OR X2>37 OR Y2<1 OR Y2>22 THEN RETURN
2570 IF FN BS(0)<>42 THEN GOTO 2600
2580 PT=PT+1:IF PT=48 THEN RETURN
2590 T=4+INT (PT/12):IF T<>NM THEN NM=T:PM(NM,1)=X2:PM(NM,2)=MY
2600 POKE 211,PM(1,1):POKE 214,PM(1,2):SYS 58640
2610 PRINT " ";
2620 POKE 211,X2:POKE 214,Y2:SYS 58640:PRINT "{GREEN}Z̲";
2630 FOR I=1 TO NM-1
2640 :PM(I,1)=PM(I+1,1)
2650 :PM(I,2)=PM(I+1,2)
2660 NEXT I
2670 PM(NM,1)=X2:PM(NM,2)=Y2
2680 MX=X2:MY=Y2
2690 RETURN
2700 FOR K=1 TO 2
2710 :MM=MM+1:IF MM>3 THEN MM=1
2720 :I=MM
2730 :TX=CO(I,1):TY=CO(I,2)
2740 :GT=((TX+3)/4)=INT ((TX+3)/4) AND ((TY+2)/3)=INT ((TY+2)/3)
2750 :IF NOT GT THEN GOTO 2776
```

```
276Ø :ON RND (1)*8+1 GOTO 277Ø,277Ø,277Ø,277Ø,277Ø,2775,2775,2776
277Ø :IF RND (1)>Ø.5 THEN GOTO 2773
2771 :IF MX<TX THEN DE(I)=3:GOTO 2776
2772 :IF MX>TX THEN DE(I)=1:GOTO 2776
2773 :IF MY<TY THEN DE(I)=2:GOTO 2776
2774 :IF MY>TY THEN DE(I)=4:GOTO 2776
2775 :DE(I)=INT (RND(1)*4)+1
2776 :X2=TX:Y2=TY:TD=DE(I)
278Ø :IF TD=1 THEN X2=X2+1
279Ø :IF TD=2 THEN Y2=Y2-1
28ØØ :IF TD=3 THEN X2=X2-1
281Ø :IF TD=4 THEN Y2=Y2+1
283Ø :IF X2<1 OR X2>37 OR Y2<1 OR Y2>22 THEN GOTO 293Ø
284Ø :POKE 211,TX:POKE 214,TY:SYS 5864Ø
285Ø :PRINT SP$(I);
286Ø :SP$(I)=FA$(FN FA(Ø)):T=FN BS(O):IF T=87 THEN T=119
287Ø :IF T=9Ø THEN T=122
289Ø :SP$(I)=SP$(I)+CHR$(T)
29ØØ :POKE 211,X2:POKE 214,Y2:SYS 5864Ø
291Ø :PRINT"{WHITE}W";
292Ø :CO(I,1)=X2:CO(I,2)=Y2
293Ø :FOR I=1 TO 3
294Ø :  TX=CO(I,1)
295Ø :  TY=CO(I,2)
296Ø :  FOR J=1 TO NM
297Ø :    IF TX=PM(J,1) AND TY=PM(J,2) THEN KL=1:GOTO 2995
298Ø :  NEXT J
299Ø :NEXT I
2995 NEXT K:RETURN
3ØØØ REM
3Ø1Ø REM *** ENDE ***
3Ø2Ø REM
3Ø3Ø PRINT "{CLR}"
3Ø4Ø PRINT "DAS SPIEL IST BEENDET !!!"
3Ø5Ø IF PT=48 THEN PRINT "SIE HABEN ALLE PUNKTE !":RETURN
3Ø6Ø PRINT "SIE HABEN";PT;"PUNKTE..."
3Ø7Ø RETURN
```

■ Programmbeschreibung für Packman

Dieses Spiel muß wohl etwas ausführlicher besprochen werden, da seine Struktur nur schwer zu erkennen ist. Es ist ziemlich verschachtelt, außerdem wird mit vielen Variablen gearbeitet, die einem das Verständnis fremder Programme bekanntlich nie einfach machen.

Zunächst werden zwei Arrays definiert. Im ersten, CO (wie COmputer) werden die Koordinaten von zwei sogenannten Packman-Fressern gespeichert. CO(1,1) und CO(1,2) stehen für den ersten Fresser, CO(2,1) und CO(2,2) für den zweiten ...

In dem Feld »SP$« werden später Daten über das Feld gespeichert, die der Fresser gerade besetzt hat. Dadurch soll gewährleistet werden, daß die Felder, über die der Packman-Fresser zieht, unverändert bleiben (funktioniert ähnlich wie bei Drachenjagd).

In Zeile 1040 wird ein Stringarray definiert, das 16 Elemente enthält. Diesmal wird das Element mit dem Index null wieder genutzt! In der Schleife der Zeilen 1050 bis 1070 werden diesem Stringarray die Zeichen für die Farben zugeordnet. Sie erhalten diese inversen Zeichen, wenn Sie nach einem Anführungszeichen jeweils [CTRL]+[ZIFFERNTASTE]

drücken. Als Zifferntasten sind die Tasten von eins bis acht gemeint, da [CTRL]+[9] keine Farbe mehr ist, sondern den Invers-Modus einschaltet. Die restlichen acht Farben erhalten Sie mit [CBM]+[ZIFFERNTASTE].

In 1100 werden Ihre gesammelten Punkte erst einmal auf null gesetzt. Die Zahl Ihrer Leben (sprich der Packmans) wird auf drei begrenzt. Hier können Sie natürlich beliebig viele Leben einsetzen. Dadurch wird das Spiel aber langweilig, wenn Sie praktisch keine Begrenzung mehr haben. Auch die Länge der Packman-Schlange wird initialisiert; sie wird auf vier gesetzt.

Die POKE-Befehle in Zeile 1130 sollten Ihnen bekannt vorkommen. Diese beiden setzen die Rahmen- und Hintergrundfarbe auf Schwarz. Anschließend wird in 1140 der Bildschirm gesetzt, der Invers-Modus und die Farbe Weiß eingestellt. Dadurch, daß jetzt noch 40 Leerzeichen folgen, wird eine gesamte Linie quer über den Bildschirm gezogen. Die nächsten paar Zeilen steuern den Bildschirmaufbau. Dafür wird eine Schleife siebenmal durchlaufen.

Betrachten Sie bitte Zeile 1150: Hinter 19 steht »STEP 3«. Das heißt, daß die Zählvariable nach jedem Durchlauf nicht um eins, sondern um drei erhöht wird. Wenn die Variable »I« also beim ersten Mal den Wert eins hat, hat sie beim zweiten Durchlauf den Wert vier, dann sieben, zehn, 13, 16 und 19. Das ergibt alles in allem sieben Durchläufe, bevor die Schleife abgebrochen wird. Zeile 1160 setzt den Cursor auf die nullte Stelle in der »I«ten Zeile. Demnach beginnt alle drei Zeilen die Ausgabe der folgenden Befehle:

Zeile 1170 schaltet den Invers-Modus ein, die Farbe wird auf Weiß gesetzt, ein Leerzeichen wird (invers!) ausgegeben; anschließend wird der Invers-Modus wieder ausgeschaltet, es folgen 38 Leerzeichen, bevor mit [CTRL]+[9] am rechten Rand noch einmal ein »Balken« gezogen wird. Da dieses Verfahren mit den »Balken« am rechten und linken Rand in allen Ausgabezeilen beibehalten wird, entsteht auf dem Bildschirm eine Umrahmung.

Sie werden sich vielleicht fragen, warum ich am Ende von Zeile 1170 (wie auch bei Zeile 1140) kein [CTRL]+[0] für Invers-aus gedrückt habe. Wird eine PRINT-Zeile mit einem Strichpunkt abgeschlossen, so wird der eventuell eingeschaltete Invers-Modus beibehalten. Für PRINT-Anweisungen ohne Strichpunkt am Ende gilt das jedoch nicht.

Die Zeile 1180 schaut aber komisch aus: Nachdem die Invers-Darstellung eingeschaltet wurde, und die Zeichenfarbe auf Weiß gesetzt wurde, kommen da hin und wieder einige [CRSR RIGHT]-Anweisungen vor. Dazu müssen Sie wissen, daß ein [SPACE] invers dargestellt einen »Block« ergibt (probieren Sie es ruhig aus!). Dagegen gibt es kein inverses [CRSR RIGHT]; diese Richtungsbewegung wird immer ausgeführt. Das heißt also, daß an dieser Stelle nichts in der Zeichenfarbe erscheinen soll, sondern wirklich eine Leerstelle ausgegeben werden soll. Die Farbe Weiß (das große, inverse E) können Sie mit [CTRL]+[2] einschalten, während das inverse Karo mit [CBM]+[7] zu erreichen ist. In Zeile 1190 sehen Sie auch gleich die Bestätigung dessen, was ich oben gesagt habe; nach einem Strichpunkt (am Ende von Zeile 1180) muß hier der Invers-Modus nicht wieder eingeschaltet werden.

Da die Zeile 1190 aber ohne Semikolon (Strichpunkt) abgeschlossen wird, muß Zeile 1200 diese Extra-Art erst wieder einschalten.

In Zeile 1220 wird die Schleife erst abgeschlossen. Wie Sie sehen, gibt die Schleife immer drei Zeilen auf einmal aus. Daher kann die Schleife auch in Dreier-Schritten erhöht werden.

Um in der untersten Zeile auch noch einmal eine freie Zeile (mit »Umrahmung«) zu erhalten, muß noch einmal eine »leere Zeile« in Zeile 1230 ausgegeben werden. Zeile 1250 zeigt schließlich drei Zeichen für die Zahl Ihrer Leben an (allerdings in einer anderen Farbe, nämlich CTRL+6 für Grün).

Zeile 1260 dimensioniert die zweidimensionale Variable PM (wie PackMan). In den Variablen PM (X,1) und PM(X,2) werden jeweils die Koordinaten eines Packmangliedes dargestellt (X steht hier für eine beliebige Zahl zwischen eins und acht). Das heißt gleichzeitig, daß die Schlange nicht beliebig lang werden kann. Sie ist auf acht Glieder begrenzt. Aber das sollte eigentlich genug sein.

In der folgenden verschachtelten Schleife werden die ganzen Nahrungspunkte auf dem Bildschirm ausgegeben. Die Zeile 1290 bestimmt eine Farbe »F$« zufällig aus fünf möglichen. MID$(,,1) bedeutet, daß ein String der Länge eins aus dem ursprünglichen String kopiert wird. Die Stelle, ab der ein Zeichen »entnommen« wird, wird eben per Zufall bestimmt. Der ursprüngliche String besteht nur aus fünf unterschiedlichen Tastenkombinationen zwischen Ziffertasten und CTRL bzw. CBM.

Um die Ausgabe (und auch die Schleifenanweisungen) richtig verstehen zu können, ist es sinnvoll, eine kurze Tabelle auszugeben, die Ihnen zeigt, wo die Nahrungspunkte verteilt sind. Gleich zum Mitlesen: Die Variable »I« ist für die Spalte, »J« für die Zeile zuständig:

```
 5/4   5/7   5/10   5/13   5/16   5/19
 9/4   9/7   9/10   9/13   9/16   9/19
 ...   ...   ....   ....   ....   ....
33/4  33/7  33/10  33/13  33/16  33/19
```

Durch diese Verteilung der Variablen kommt es zustande, daß die Ausgabe der Nahrungspunkte von links nach rechts erfolgt und nicht von oben nach unten.

Wenn Sie die Werte einmal durchzählen, werden Sie feststellen, daß »I« achtmal und »J« sechsmal durchlaufen wurde (achten Sie bitte auf STEP 3!). Das heißt, daß insgesamt 48 Nahrungspunkte auf dem Bildschirm stehen sollten. Außerdem sollten sie auch noch in fünf unterschiedlichen Farben »glitzern«.

Zum Abschluß der Vorbereitungen werden noch zwei Funktionen definiert, die (wie schon früher) das Zeichen bzw. die Farbe an einer bestimmten Stelle des Bildschirms definieren. Daher die Namen »BS« (für BildSchirm) und »FA« (für FArbe).

Kommen wir jetzt zum Programmteil »SPIEL«. Das dürfte ein harter Brocken werden. Aber gemeinsam werden wir es schaffen! Sollten Sie zwischendurch die Lust verlieren, laden Sie das Programm einfach und spielen ein, zwei Runden. Das baut Sie hoffentlich wieder etwas auf.

In den ersten drei Zeilen wird das zweidimensionale Array »CO«, das in Zeile 1030 definiert wurde, mit Werten belegt. Wie gesagt, enthält dieses Feld die X- und Y-Koordinaten der Packman-Fresser. Daher wurden die Zuweisungen in den Zeilen 2030 bis 2050 paarweise angeordnet.

Glauben Sie aber bitte nicht, diese »Vorbelegung« in dem Programmteil »VORBEREITUNGEN« vorverlegen zu können. Diese Werte werden noch öfters benötigt, zum Beispiel wenn Sie ein Leben verloren haben, und der Bildschirm wieder auf die Ausgangswerte zurückgesetzt wird. Wie Sie später noch sehen werden, erfolgt da ein Sprung auf diese Zeilen (von 2410 aus). Wofür die Variable DE(I) gut ist, werden Sie später noch sehen.

Die Schleife in den Zeilen 2070 bis 2110 dürfte ziemlich klar sein: Sie bringt die Packman-Fresser auf den Bildschirm. Dazu werden einfach die Inhalte der Variablen CO als Cursor-Positionen verwendet, an die der Cursor direkt plaziert werden kann (Zeile 2080). An diese Stellen werden jeweils weiße »Kringel« gezeichnet (zu erreichen über [SHIFT]+[W]). Anschließend wird noch die Variable »SP$« für den entsprechenden Packman als ein Leerzeichen festgelegt. »SP$« enthält bekanntlich den Wert, den der Packman-Fresser von einem Feld vertrieben hat (siehe oben). Hier wird demnach festgelegt, daß die Felder der Fresser »vor« dem Anfang keinen besonderen Wert aufweisen.

Die nächste Schleife weist dem Packman seine Koordinaten zu (Zeilen 2120 bis 2150). Die Zeile bleibt bei 22, nur die X-Koordinate (Spaltenzahl innerhalb dieser Zeile) ändert sich. Die Schlange nimmt die Spalten von eins bis zu seiner Länge ein; diese Länge ist in »NM« gespeichert (siehe 2120).

In »MX« und »MY« werden explizit noch einmal die Koordinaten des ersten Packman-Gliedes gespeichert. Dies wird für spätere Berechnungen wichtig. Wiederum eine Schleife ist es, die den Packman »zu Papier« bringt. Sie sehen also, daß die Schleifenkonstruktion oft verwendet wird.

Die Zeile 2180 »POKEt« an die Speicherstellen des Cursors wieder einfach die Koordinaten, die in den Variablen PM(I,1) und PM(I,2) gespeichert sind (analog zu den Packman-Fressern). Insofern dürfte diese Schleife keine Probleme mehr bereiten.

Die Variablen »PR«, »KL« und »MM« werden initialisiert. Dazu möchte ich Ihnen aber noch die Bedeutung dieser Variablen erklären. »PR« speichert die Richtungsänderung Ihres Packman (Packman Richtung). Dadurch wird gewährleistet, daß sich Ihr Packman immer in derselben Richtung weiterbewegt, auch wenn Sie keine Taste gedrückt haben. »KL« (wie KoLlision) gibt an, ob eine Kollision zwischen einem Fresser und Ihnen stattgefunden hat. »MM« schließlich ist eine Art Zählvariable, auf die ich später noch genauer eingehen werde.

In Zeile 2220 wird die Zahl der Packman um eins verringert, damit das Spiel auch irgendwann ein Ende findet. Sollte die Zahl Ihrer Packman bereits unter null gesunken sein, muß das Spiel sofort abgebrochen werden.

Ist bis hierher alles in Ordnung, kann die Meldung <<< FERTIG >>> auf dem Bildschirm ausgegeben werden (2250 und 2260).

Der Sprung von 2270 nach 2480 ist für die Steuerung des Packman verantwortlich. Ab 2480 steht nämlich die Routine, die bei einem eventuellen Tastendruck Ihrerseits die Richtung Ihrer Schlange verändert bzw. die Orientierung beibehält, falls keine Taste gedrückt wurde. Ich schlage vor, wir betrachten uns dieses Unterprogramm näher, bevor wir uns mit dem weiteren Verlauf des Hauptteils beschäftigen.

In »GT« (wie GeTroffen) wird festgehalten, ob der Anfang Ihrer Schlange einen Nahrungspunkt getroffen hat. Wenn Sie sich vielleicht noch einmal die oben aufgeführte Tabelle der Koordinaten der Nahrungspunkte anschauen und dann mit unserer Formel vergleichen, werden Sie folgendes feststellen: Addiert man zu einem X-Wert (jeweils der ersten Zahl) drei, erhält man ein Vielfaches von vier. Daher gilt für diese X-Koordinaten: (X+3)/4=INT((X+3)/4).

Für »X« wird in unserer Formel »MX« eingesetzt, das ist der X-Wert des Schlangenkopfes (siehe auch Zeile 2160). So ähnlich funktioniert das mit der Y-Achse: Sobald Sie zu einem Wert zwei addieren, ist die Summe durch drei teilbar: (Y+2)/3=INT((Y+2)/3). Auch hier wird für »Y« »MY« eingesetzt. Wenn beide Bedingungen erfüllt sind, heißt das, daß der Kopf des Packman einen Nahrungspunkt gefressen hat. Das »AND« ist dafür verantwortlich, daß diese Aussage nur dann als wahr gilt, wenn wirklich beide Bedingungen erfüllt sind. Aber was heißt wahr?

Bei der IF-Abfrage haben wir meistens zwei Zahlen oder zwei Buchstaben (auch in Form von Variablen) miteinander verglichen. Dabei haben wir gesagt, daß die Bedingung als erfüllt gilt, wenn die beiden Zahlen zum Beispiel gleich waren.

Beispiel:

```
IF 3=5 THEN ...
```

Der Computer berechnet das Ganze aber anders. Er berechnet die linke Seite des Gleichheitszeichens (da könnte ja ein mathematischer Term oder sonst etwas Schwieriges stehen), dann die rechte Seite. Daraufhin vergleicht er diese beiden. Dieser Vergleich führt zu dem Ergebnis »wahr« oder »falsch«, je nach Bedingung. Die obige Abfrage beispielsweise führt zu »falsch«.

```
A$="J":IF A$="J" THEN ...
```

Diese Berechnung ergibt den Wert »wahr«. Dagegen:

```
A$="N":IF A$="J" THEN ...
```

ergibt »falsch«. Jetzt kann man so eine Abfrage auch auseinanderziehen. Die oben genannten Beispiele lassen sich umformulieren in:

```
A$="J":AF=A$="J"
IF AF THEN ...
```

Lassen Sie sich nicht verwirren! Vielleicht ist die Schreibweise »AF=(A$="J")« einsichtiger. Der Variablen »AF« (wie AbFrage) wird der Wert des Vergleichs zwischen »A$« und »J« zugewiesen. Damit Sie mir das Ganze auch glauben und etwas herumexperimentieren können, schlage ich vor, Sie geben folgendes kurze Programm ein:

```
10 A$="J"
20 AF=A$="J"
30 IF AF THEN PRINT "GLEICH!"
```

Lassen Sie es einmal laufen. Erstaunt? Aber geben Sie jetzt doch einmal

```
PRINT AF
```

ein. Das gibt Ihnen den Wert »-1« aus. Wenn Sie jetzt die Bedingung in Zeile 20 ändern, meinetwegen in:

```
20 AF=A$="N"
```

und jetzt das Programm ein zweites Mal laufenlassen, können Sie sich anschließend mit PRINT AF wieder den Inhalt dieser Variablen ansehen. Diesmal ist er null!

Wir können jetzt also festhalten, daß ein Vergleich:

- entweder das Ergebnis WAHR (-1)

- oder das Ergebnis FALSCH (0)

hat. Wenn Sie sich erinnern: Wir haben früher schon Variablen so getestet: IF A THEN ...

Damals wurde festgestellt, daß die Bedingung als wahr erkannt wird, sobald die Variable ungleich null ist. Sehen Sie die Zusammenhänge? Kommen wir aber mal wieder zu unserem Programm zurück. Wir waren bei der Zeile 2480 hängengeblieben. Wir sind noch nicht ganz fertig mit dieser Zeile:

Sollte also »MX« ein Vielfaches von drei sein, ist die Bedingung zwischen »FL« und dem »AND« erfüllt. Ist die Bedingung rechts von »AND« als »wahr« anerkannt, ist »MY« ein Vielfaches von drei. Nur wenn beide Male »wahr« herauskommt, darf die Variable »GT« auf »wahr« gesetzt werden, und dafür sorgt das »AND«.

In Zeile 2490 kommt aber schon wieder etwas Neues: »IF NOT FL THEN ...« Das heißt, daß die Abfrage genau umgekehrt wird. Was vorher als falsch abgetan wurde, wird mit einem »NOT« davor zu wahr und umgekehrt. Das heißt also, wenn FL=0 (für falsch) ist, gilt die Bedingung in 2490 erfüllt, das Programm springt nach Zeile 2510. Das ist gleichbedeutend mit einer Tastatursperre, da somit keine Eingabe angenommen wird; das wiederum bedeutet, daß Richtungsänderungen nur an den Kreuzungen vorgenommen werden können. Dadurch wird vermieden, daß Sie quer über den Bildschirm laufen können.

In Zeile 2500 wird die Tastatur abgefragt. Liegt keine Eingabe Ihrerseits vor, verzweigt das Programm nach 2510, um die alte Richtung beizubehalten. War die festgestellte Taste ein

[K], [A], [J] oder [Z] kann dementsprechend eine Richtungsänderung vorgenommen werden (»PR« wie Packman Richtung wird verändert).

Auf alle Fälle trifft der Interpreter auf die Zeile 2510, egal ob Sie eine beliebige, eine erlaubte oder gar keine Taste gedrückt haben. Da werden zunächst (auch wieder für die Überprüfung) zwei neuen Variablen »X2« und »Y2« die aktuellen Koordinaten übergeben. Daraufhin wird in den Zeilen 2520 bis 2550 »PR« ausgewertet. Je nach Richtung werden die »Zweitkoordinaten« »X2« und »Y2« verändert.

In Zeile 2560 wird das Ergebnis dieser »Berechnungen« überprüft. Liegen die neuen Werte außerhalb des Labyrinths, wird die Unterroutine abgebrochen, der C64 kehrt zu Zeile 2270 zurück. Waren die neuen Koordinaten erlaubt, wird geprüft, was für ein Zeichen auf dem zukünftigen Feld liegt. Ergibt die Funktion BS für diese Stelle etwas anderes als 42 (für Sternchen = Nahrungspunkt), verzweigt der Interpreter zur Zeile 2600.

Die Zeilen 2580 und 2590 werden demnach nur behandelt, wenn Sie mit diesem Schritt einen Nahrungspunkt fressen werden. Zunächst wird Ihr Punktekonto um eins erhöht. Sollten Sie bereits 48 Punkte verbucht haben, haben Sie alle Nahrungspunkte schon aufgefressen. Das heißt, das Programm muß abgebrochen werden.

Andernfalls wird überprüft, ob Ihre Schlange nicht länger werden müßte. Nach jeweils zwölf Punkten bekommen Sie ein Körperglied dazu. Dafür wird die Variable »T« eingeführt. Zur »Grundlänge« vier wird das Ergebnis der Division »PT/12« addiert. Sollte »T« von der bisherigen Länge »NM« abweichen, wird »NM« auf diesen neuen Wert gesetzt. Die Koordinaten des Schlangenkopfes bleiben gleich. Sie müssen aber dem neuen Glied zugewiesen werden.

Die Zeilen ab 2600 werden jetzt wieder in allen Fällen behandelt, gleichgültig ob Sie einen Nahrungspunkt erwischt haben oder nicht. Hier wird der Cursor zunächst auf die Koordinaten des letzten Körperglieds der Schlange gesetzt. Durch das »SPACE« in 2610 wird genau dieser Teil gelöscht. In 2620 wird aber wieder ein Ausgleich geschaffen, da dort der Cursor auf die neu berechneten Koordinaten gesetzt wird. Anschließend wird an dieser Stelle ein »Packman-Teil« ausgegeben.

In der anschließenden Schleife werden die Koordinaten der einzelnen Glieder um jeweils eins »nach vorne geschoben«, indem den Variablen die Werte Ihrer »Vorgänger« (im wahrsten Sinne des Wortes) zugewiesen werden.

Um die Schlange vollends auf den neuesten Stand zu bringen, werden dem Kopf die aktuellen Koordinaten, nämlich »X2« und »Y2« zugewiesen. Damit der Computer auch weiß, wo sich dieser Kopf befindet, müssen noch »MX« und »MY« modifiziert werden. Damit hätten wir es aber fürs erste.

Der Computer ist nach dem »RETURN« in Zeile 2690 (eventuell auch früher) in Zeile 2280 angekommen. Dort findet er die Abfrage, ob bereits alle Punkte gefressen worden sind. Die Abfrage kennen wir bereits aus Zeile 2580; dort konnte aber erst die Routine zur

Steuerung des Packman verlassen werden. Mit diesem RETURN würde das Programm zum Hauptprogramm (nämlich Zeile 90) zurückkehren.

Die Zeile 2290 ruft die Routine ab 2700 auf, die für die Steuerung der Packman-Fresser zuständig ist. Dieses Unterprogramm beginnt gleich mit einer Schleife, die besagt, daß jeweils nur zwei der drei Fresser bewegt werden sollen. Daher benötigt man noch eine zweite Variable (hier »MM«), die einem mitteilt, welche zwei Figuren gerade an der Reihe sind. Diese Variable wird um eins pro Schleifendurchlauf erhöht. Sollte sie die zulässige Anzahl von drei überschreiten, muß sie wieder auf eins gesetzt werden, da wir sonst bald keine Packman-Fresser mehr zur Verfügung hätten.

Die Variable »I« wird rein aus Tippfaulheit eingeführt. Dadurch spart man sich bei jedem Vorkommen von »MM« einen Buchstaben; man hat zwar am Anfang einen kleinen Mehraufwand, der sich aber bald amortisiert. Auch um Tipparbeit zu sparen und so lange Programmzeilen erst zu ermöglichen, werden »TX« und »TY« anstelle von CO(I,1) und CO(I,2) bzw. sogar CO(MM,1) und CO(MM,2) verwendet.

Das Argument mit der Länge einer Programmzeile ist nicht aus der Luft gegriffen, wie Sie in Zeile 2740 sehen. Stellen Sie sich vor, Sie müßten jedes »TX/TY« durch »CO(...)« ersetzen. Das wäre ausgeschlossen. Über den Aufbau dieser (und der nächsten) Zeile lasse ich mich hier nicht noch einmal aus. Lesen Sie dazu bitte gegebenenfalls die Ausführungen weiter oben noch einmal durch. Prinzipiell wird auch hier wieder überprüft, ob sich einer der Fresser an einem Nahrungspunkt befindet. Wenn ja, darf er seine Richtung ändern, wenn nicht, muß er sie beibehalten. In dem Fall erfolgt ein Sprung nach Zeile 2776 (Zeile 2750).

Für den Fall, daß der Fresser seine Richtung ändern darf, müssen verschiedene Möglichkeiten in Betracht gezogen werden. Dieses Programm bietet vier Variationen: Entweder wird bei allen vier Koordinaten geprüft, in welcher Richtung vom aktuellen Standpunkt aus sich Ihr Packman befindet; oder es werden nur zwei dieser vier Koordinaten dafür herangezogen; oder die Richtung wird per Zufall gesteuert; oder aber der Fresser behält seine Richtung bei. Wie kommt es jetzt zu diesen Unterscheidungen? Betrachten Sie bitte Zeile 2760: Zunächst wird eine Zufallszahl (zwischen eins und acht) bestimmt; von dieser Zahl hängt es ab, welcher Weg eingeschlagen werden soll. In fünf von acht Fällen springt das Programm zu Zeile 2770. In weiteren zwei Fällen wird nach 2775 verzweigt (für völligen Zufall) und nur in einem Fall kommt das Programm nach 2776, ohne eine neue Richtung bestimmt zu haben.

Gehen wir aber noch einmal zurück zu Zeile 2770. In fünf von acht Fällen landet hier der C64. Jetzt wird aber noch einmal unterschieden: Ist eine Zufallszahl zwischen null und eins größer als 0.5, werden nur zwei Richtungsänderungen in Betracht gezogen. Gehen wir das Ganze kurz noch einmal mit konkreten Zahlen durch:

Angenommen, der Zufallsgenerator bringt in Zeile 2760 die Eins. Dann springt der Computer nach 2770. Vorausgesetzt, die zweite Zufallszahl liegt zwischen null und 0.5, sind die Fälle in den Zeilen 2771 bis 2774 möglich. Liegt der Wert dieser zweiten Zufallszahl aber

zwischen 0.5 und eins, kommen nur noch die Abfragen in 2773 und 2774 in Betracht. Das gleiche gilt, wenn die erste Zufallszahl eine Zwei, Drei, Vier oder Fünf war.

Bei Sechs oder Sieben springt das Programm nach 2775 und bei Acht nach 2776. Jetzt gilt es noch herauszufinden, was das Programm in den einzelnen Fällen macht. Bei 2771 bis 2774 werden Ihre Koordinaten mit denen des gerade bearbeiteten Fressers verglichen. Abhängig von diesem Vergleich werden die Richtungen bestimmt. Diese Richtungen werden in »DE(I)« gespeichert.

Bei reinem Zufall wird der Variablen DE(I) eine Zahl zwischen eins und vier zugewiesen. In 2776 wird »X2« und »Y2« die aktuellen Koordinaten des »Killers« zugewiesen. TD schließlich übernimmt den Wert von »DE(I)«. Jetzt wissen Sie auch, wofür die Zeile 2060 gut war. Sollte beim ersten Durchlauf nämlich keine Richtungsveränderung bei einem der drei Fresser auftreten, wird die »alte« Richtung weiterverwendet. Darum muß die Variable »DE(I)« schon vor dem ersten Lauf mit sinnvollen Werten belegt sein!

Je nach aktueller Richtung »TD« werden die »Zweitkoordinaten« X2 und Y2 verändert. Auch hier muß geprüft werden, ob sich die Killer vielleicht außerhalb des Spielfeldes bewegen? Sobald sie Gefahr laufen, aus dem Feld zu kommen, wird nach 2930 verzweigt; damit wird unter anderem vermieden, daß die richtigen Koordinaten (CO(I,1) und CO(I,2)) auf diese sinnlosen Werte gesetzt werden (das passiert bereits in 2920).

Ist alles in Ordnung, wird der Cursor auf die »alten« Koordinaten gesetzt und der ursprüngliche Wert dieses Feldes ausgegeben (gespeichert in SP$(I)).

Jetzt kann »SP$(I)« neu belegt werden; und das geschieht gleich in 2860, bevor noch der Wert des neuen Feldes vom Fresser überschrieben werden kann. Dazu wird mittels der Funktion »FA(0)« der Farbwert an dieser Stelle ermittelt. Wenn man dieses Ergebnis als Index für »FA$« verwendet, erhält man den Farbwert in Form eines Strings (siehe auch Zeilen 1080 und 1090).

»T« wird der Inhalt des Bildschirmspeichers an der neuen Stelle zugeordnet. Dadurch erhält man den Code eines Zeichens für POKE. Da das Programm aber mit der Funktion CHR$ arbeiten will (wie Sie später noch sehen werden), muß der Code für POKE in einen ASCII-Code umgewandelt werden. Wenn Sie sich ANHANG E und ANHANG F in Ihrem C64-Handbuch ansehen, können Sie diese Umwandlung von Hand vornehmen, wie ich es auch getan habe.

Um nach dem Verlassen ein Feld wieder restaurieren zu können, muß man das entsprechende Zeichen in der richtigen Farbe ausgeben. Dazu verbindet man die Farbinformation (in SP$(I)) mit dem Zeichen CHR$(T) zu einem neuen String »SP$(I)« (Zeile 2890).

Nachdem der Inhalt des neuen Feldes so gerettet wurde, kann es jetzt endlich beschrieben werden. Der Cursor wird in 2900 auf die neuen Werte gesetzt, und 2910 gibt das Zeichen für den Killer aus. Anschließend werden dessen Koordinaten ebenfalls auf den neuesten Stand gebracht.

Ab Zeile 2930 findet noch die Überprüfung statt, ob es mittlerweile eine Kollision zwischen Ihnen und einem Fresser gegeben hat. Der einzige Hinweis, den ich Ihnen hier noch gebe, ist, daß »KL« für KoLlision steht.

Nachdem diese Kontrollschleife erfolgt ist, wird noch die äußerste Schleife geschlossen, das heißt, jetzt wird das Ganze noch einmal für den zweiten Fresser durchgeackert.

Irgendwann einmal wird es das Programm aber hinter sich gebracht haben, und dann steht der C64 in der Zeile 2300. Dort wird zunächst einmal geprüft, ob eine Kollision stattgefunden hat. Wenn nicht, kann ohne Komplikationen wieder zur Packman-Steuerung gesprungen werden. Andernfalls müssen alle möglichen Daten zurückgesetzt werden, und das kommt jetzt:

In der ersten Schleife (Zeilen 2310 bis 2340) werden alle drei Packman-Fresser vom Bildschirm gelöscht, indem ihre momentanen Positionen mit den ursprünglichen Inhalten »SP$(I)« überschrieben werden. Anschließend wird auch die gesamte Packman-Schlange gelöscht. Zu guter Letzt wird der Cursor noch in die vorletzte Bildschirmzeile gesetzt (Zeile 2390), um dort eines der Lebenszeichen zu löschen. Wie das genau funktioniert, sollten Sie herausfinden. Schließlich wissen Sie ja, was in »NM« gespeichert ist, also ...

Ich gebe zu, das war beileibe keine kurze Anleitung. Aber ich glaube, daß das Programm auch nicht so einfach zu verstehen war. Also, es hat sich rentiert!

16 Labyrinth

Einführung

Dieses Programm ähnelt wieder dem vorletzten, der Drachenjagd. Aber keine Sorge, es kommen wieder völlig neue Spielelemente vor, es dürfte also keine Langeweile geben.

Sobald Sie das Spiel gestartet haben, erscheint auf dem Bildschirm ein Gitter. Dieses Gitter wird immer mehr zerteilt, bis am Schluß ein Labyrinth daraus wird. Dabei achtet das Programm darauf, daß es einen Weg durch das Gänge-Gewirr gibt; es kann also nicht passieren, daß Sie in einem Gang von der Umwelt völlig abgeschlossen stehen und nicht das Ziel erreichen.

Sobald die Gänge gezeichnet worden sind, verschwindet dieser Lageplan, und Sie sehen nur noch die Umrandung, den Aufenthaltsort des Drachen und die Stelle, an der Sie gerade stehen. Ihre Figur können Sie mit den Tasten:

[K] nach rechts,
[J] nach links,
[I] nach oben und
[M] nach unten steuern.

Diese Tastenbelegung wird während des Spiels auch auf dem Bildschirm dargestellt. Sollten Sie während Ihrer Irrfahrt auf eine Mauer stoßen, wird diese angezeigt. Sie können sich allmählich durch das Labyrinth vortasten, wenn da nicht dieser Drache wäre ...

Sobald der Sie nämlich erwischt, ist es um Sie geschehen. Auf gut deutsch, das Spiel ist dann beendet. Ich wünsche Ihnen natürlich viel Erfolg!

Listing zu Labyrinth

```
10 REM *****************
20 REM ***           ***
30 REM *** LABYRINTH  ***
40 REM ***           ***
50 REM *****************
60 REM
70 GOSUB 1000
80 GOSUB 2100
90 GOSUB 3000
100 END
1000 REM
1010 REM *** VORBEREITUNGEN ***
1020 REM
1030 DIM MA(19,11,4)
1040 DEF FN R(X)=INT (RND(1)*X)+1
1050 RX=FN R(19):RY=FN R(11)
1060 PRINT "{CLR}";
1070 PRINT "{RVSON,GREY 2,39SPACE}"
1080 FOR I=1 TO 11
1090 :PRINT "{RVSON,SPACE,RVOFF,SPACE,RVSON,SPACE,RVOFF,SPACE,RVSON,SPAC
     E,RVOFF,SPACE,RVSON,SPACE}";
1100 :PRINT "{RVOFF,SPACE,RVSON,SPACE,RVOFF,SPACE,RVSON,SPACE,RVOFF,SPAC
     E,RVSON,SPACE,RVOFF}";
1110 :PRINT "{SPACE,RVSON,SPACE,RVOFF,SPACE,RVSON,SPACE,RVOFF,SPACE,RVSO
     N,SPACE,RVOFF,SPACE}";
1120 :PRINT "{RVSON,SPACE,RVOFF,SPACE,RVSON,SPACE,RVOFF,SPACE,RVSON,SPAC
     E,RVOFF,SPACE,RVSON,SPACE}";
1130 :PRINT "{RVOFF,SPACE,RVSON,SPACE,RVOFF,SPACE,RVSON,SPACE,RVOFF,SPAC
     E,RVSON,SPACE,RVOFF}";
1140 :PRINT "{SPACE,RVSON,SPACE,RVOFF,SPACE,RVSON,SPACE,RVOFF,SPACE,RVSO
     N,SPACE}"
1150 :PRINT "{RVSON,39SPACE}"
1160 NEXT I
1170 PRINT "{BLACK,3SPACE}<<< ICH BAUE NUN DAS LAGER AUF >>>"
1180 CN=1:C=0
1190 R=1:L=1
1200 O=1:U=1
1210 IF RX<19 THEN R=-(MA(RX+1,RY,0)>0)
1220 IF RX>1  THEN L=-(MA(RX-1,RY,0)>0)
1230 IF RY<11 THEN U=-(MA(RX,RY+1,0)>0)
1240 IF RY>1  THEN O=-(MA(RX,RY-1,0)>0)
1250 C=L+R+O+U
1260 IF C=4 OR (C>2 AND FN R(10)<3) THEN GOTO 1720
1270 ON FN R(4) GOTO 1300,1400,1500,1600
1300 IF R THEN GOTO 1270
1310 POKE 211,RX+RX:POKE 214,RY+RY-1:SYS 58640
1320 PRINT "{RVOFF,SPACE}";
1330 MA(RX,RY,0)=MA(RX,RY,0)+1
1340 MA(RX,RY,1)=1
1350 RX=RX+1
1360 MA(RX,RY,3)=1
1370 MA(RX,RY,0)=MA(RX,RY,0)+1
1380 GOTO 1700
1400 IF U THEN GOTO 1270
1410 POKE 211,RX+RX-1:POKE 214,RY+RY:SYS 58640
1420 PRINT "{RVOFF,SPACE}";
1430 MA(RX,RY,0)=MA(RX,RY,0)+1
1440 MA(RX,RY,2)=1
1450 RY=RY+1
1460 MA(RX,RY,4)=1
1470 MA(RX,RY,0)=MA(RX,RY,0)+1
1480 GOTO 1700
```

```
1500 IF L THEN GOTO 1270
1510 POKE 211,RX+RX-2:POKE 214,RY+RY-1:SYS 58640
1520 PRINT "{RVOFF,SPACE}";
1530 MA(RX,RY,0)=MA(RX,RY,0)+1
1540 MA(RX,RY,3)=1
1550 RX=RX-1
1560 MA(RX,RY,1)=1
1570 MA(RX,RY,0)=MA(RX,RY,0)+1
1580 GOTO 1700
1600 IF O THEN GOTO 1270
1610 POKE 211,RX+RX-1:POKE 214,RY+RY-2:SYS 58640
1620 PRINT "{RVOFF,SPACE}";
1630 MA(RX,RY,0)=MA(RX,RY,0)+1
1640 MA(RX,RY,4)=1
1650 RY=RY-1
1660 MA(RX,RY,2)=1
1670 MA(RX,RY,0)=MA(RX,RY,0)+1
1700 IF MA(RX,RY,0)=1 THEN CN=CN+1:IF CN=209 THEN GOTO 1800
1710 GOTO 1190
1720 RX=FN R(19):RY=FN R(11)
1730 IF MA(RX,RY,0)<>0 THEN GOTO 1190
1740 RX=RX+1:IF RX>19 THEN RX=1:RY=RY+1:IF RY>11 THEN RY=1
1750 GOTO 1730
1800 PRINT "{CLR,WHITE}";
1810 PRINT "{RVSON,40SPACE}";
1820 FOR I=1 TO 10
1830 :PRINT "{RVSON,SPACE,RVOFF,38SPACE,RVSON,SPACE}";
1840 :PRINT "{RVSON,SPACE,RVOFF,38SPACE,RVSON,SPACE}";
1850 NEXT I
1860 PRINT "{RVSON,SPACE,RVOFF,38SPACE,RVSON,SPACE}";
1870 PRINT "{RVSON,40SPACE}"
1880 PX=1:PY=FN R(11)
1890 POKE 211,PX+PX-1:POKE 214,PY+PY-1:SYS 58640
1900 PRINT"{RVOFF,GREY 2}&";
1910 WY=FN R(11)
1920 POKE 211,38:POKE 214,WY+WY-1:SYS 58640
1930 PRINT "{BLACK,RVSON,SPACE}";
1940 MX=19:MY=WY
1950 POKE 211,37:POKE 214,MY+MY-1:SYS 58640
1960 PRINT "{PURPLE}&";
1970 FOR I=1 TO 19
1980 :FOR J=1 TO 11
1990 : MA(I,J,0)=0
2000 :NEXT J
2010 NEXT I
2020 POKE 211,2:POKE 214,24:SYS 58640
2030 PRINT "{RVSON,LIG.BLUE,SPACE}J=LINKS{2SPACE}K=RECHTS{2SPACE}I=AUF{2
     SPACE}M=AB ";
2040 RETURN
2100 REM
2110 REM *** SPIEL ***
2120 REM
2130 GET TA$
2150 IF TA$="" THEN GOTO 2130
2160 IF TA$="K" THEN GOTO 2250
2170 IF TA$="M" THEN GOTO 2300
2180 IF TA$="J" THEN GOTO 2350
2190 IF TA$="I" THEN GOTO 2400
2200 GOTO 2500
2250 IF MA(PX,PY,1) THEN X2=PX+1:Y2=PY:GOTO 2450
2260 FOR K=-2 TO 0
```

```
2270 :POKE 211,PX+PX:POKE 214,PY+PY+K:SYS 58640
2280 :PRINT "{RVSON,WHITE,SPACE}"
2290 NEXT K:GOTO 2500
2300 IF MA(PX,PY,2) THEN X2=PX:Y2=PY+1:GOTO 2450
2310 FOR K=-2 TO 0
2320 :POKE 211,PX+PX+K:POKE 214,PY+PY:SYS 58640
2330 :PRINT "{RVSON,WHITE,SPACE}"
2340 NEXT K:GOTO 2500
2350 IF MA(PX,PY,3) THEN X2=PX-1:Y2=PY:GOTO 2450
2360 FOR K=-2 TO 0
2370 :POKE 211,PX+PX-2:POKE 214,PY+PY+K:SYS 58640
2380 :PRINT "{RVSON,WHITE,SPACE}"
2390 NEXT K:GOTO 2500
2400 IF MA(PX,PY,4) THEN X2=PX:Y2=PY-1:GOTO 2450
2410 FOR K=-2 TO 0
2420 :POKE 211,PX+PX+K:POKE 214,PY+PY-2:SYS 58640
2430 :PRINT "{RVSON,WHITE,SPACE}"
2440 NEXT K:GOTO 2500
2450 POKE 211,PX+PX-1:POKE 214,PY+PY-1:SYS 58640
2460 PRINT "{RVOFF,SPACE}";
2470 PX=X2:PY=Y2
2480 POKE 211,PX+PX-1:POKE 214,PY+PY-1:SYS 58640
2490 PRINT "{RVOFF,GREY 2}Q";
2500 IF PX=19 AND PY=WY THEN WI=-1:RETURN
2510 IF PX<>MX OR PY<>MY THEN GOTO 2560
2520 :POKE 211,MX+MX-1:POKE 214,MY+MY-1:SYS 58640
2530 :PRINT "{RVOFF,RED}Z";
2540 :WI=0
2550 :RETURN
2560 ON FN R(4) GOTO 2570,2580,2590,2600
2570 IF MX<PX THEN GOTO 2620
2580 IF MY<PY THEN GOTO 2660
2590 IF MX>PX THEN GOTO 2700
2600 IF MY>PY THEN GOTO 2740
2610 GOTO 2570
2620 IF MX=19 THEN GOTO 2660
2630 IF MA(MX,MY,0)>5 THEN MA(MX,MY,0)=0:GOTO 2650
2640 IF MA(MX,MY,1)=0 THEN GOTO 2660
2650 X2=MX+1:Y2=MY:GOTO 2800
2660 IF MY=11 THEN GOTO 2700
2670 IF MA(MX,MY,0)>5 THEN MA(MX,MY,0)=0:GOTO 2690
2680 IF MA(MX,MY,2)=0 THEN GOTO 2700
2690 X2=MX:Y2=MY+1:GOTO 2800
2700 IF MX=1 THEN GOTO 2740
2710 IF MA(MX,MY,0)>5 THEN MA(MX,MY,0)=0:GOTO 2730
2720 IF MA(MX,MY,3)=0 THEN GOTO 2740
2730 X2=MX-1:Y2=MY:GOTO 2800
2740 IF MY=1 THEN GOTO 2620
2750 IF MA(MX,MY,0)>5 THEN MA(MX,MY,0)=0:GOTO 2770
2760 IF MA(MX,MY,4)=0 THEN GOTO 2620
2770 X2=MX:Y2=MY-1
2800 POKE 211,MX+MX-1:POKE 214,MY+MY-1:SYS 58640
2810 PRINT "{RVOFF,SPACE}";
2820 MX=X2:MY=Y2
2830 POKE 211,MX+MX-1:POKE 214,MY+MY-1:SYS 58640
2840 PRINT "{RVSON,RED}Z";
2850 MA(MX,MY,0)=MA(MX,MY,0)+1
2860 IF PX=19 AND PY=WY THEN WI=-1:RETURN
2870 IF PX=MX AND PY=MY THEN WI=0:RETURN
2880 GOTO 2130
3000 REM
3010 REM *** ENDE ***
3020 REM
3030 IF WI THEN GOTO 3060
```

```
3040 PRINT "{CLR,LIG.BLUE,3DOWN}LEIDER HAT SIE DER DRACHE ERWISCHT."
3050 RETURN
3060 PRINT "{CLR,LIG.BLUE,3DOWN}SIE HABEN GEWONNEN."
3070 RETURN
```

■ Programmbeschreibung für Labyrinth

In Zeile 1030 sehen Sie zum ersten Mal ein dreidimensionales Feld. Es enthält 19*11*5 Elemente (1045). Ich will Ihnen das erklären: Die ersten zwei Indizes beginnen in unserem Fall bei eins und hören bei 19 bzw. elf auf. Beim dritten Index dagegen wird von null bis vier gezählt, macht allein für den dritten Index jeweils fünf Elemente.

Sie können mit diesem Feld genauso wie bisher mit zweidimensionalen Feldern arbeiten. Die nächste Zeile dürfte Ihnen allmählich bekannt vorkommen. In Zeile 1050 wird die Funktion auch gleich benutzt. Sie soll für »RX« einen beliebigen Wert zwischen eins und 19, für »RY« einen zwischen eins und elf liefern (siehe auch Dimensionen des Arrays MA!).

Die Zeilen 1060 bis 1170 sind für die Bildschirmsteuerung zuständig. Zeile 1060 löscht den Bildschirm, und 1070 setzt sofort eine inverse Zeile in die oberste Bildschirmzeile (siehe auch Strichpunkt in 1060!). Vielleicht wundern Sie sich über die Laufvariable »I«? Sie soll nur bis elf zählen?

Betrachten Sie sich dazu bitte die Zeilen 1090 bis 1150. Sie können gut erkennen, daß hier zwei Zeilen jeweils beschrieben werden. Das erste (einfachere) Kennzeichen ist, daß sowohl in 1140 als auch in 1150 kein Strichpunkt am Ende steht. Die andere Möglichkeit, so etwas herauszufinden, wäre die, daß Sie alle ausgegebenen Zeichen abzählen. Wenn Sie in 1090 beginnen, müssen Sie eines beachten. Die Steuerzeichen dürfen Sie nicht mitzählen. Sie verändern jeweils etwas an der Ausgabe (schalten beispielsweise den Invers-Modus ein), bewegen den Cursor aber kein bißchen. Daher dürften Sie in den ersten sechs Zeilen nur jedes zweite Zeichen zählen (praktisch nur die »SPACEs«). Und wenn Sie das durchhalten, werden Sie auf 39 kommen, entspricht also (bis auf ein Zeichen) der Länge einer Bildschirmzeile. Und die zweite Ausgabezeile (1150) besteht praktisch nur aus 40 Leerzeichen.

Die anschließende Meldung brauche ich wohl nicht besonders zu erwähnen? Jetzt beginnt allmählich der Ernst des Spiels: Zunächst stehen sechs Initialisierungen ins Haus. Alle sechs Variablen werden für den Aufbau des Labyrinths benötigt. CN ist eine Variable, die mitzählt, ob es bereits einen Weg durch das Labyrinth gibt. R, L, O und U werden gebraucht, um festzustellen, ob das Feld neben dem gerade untersuchten schon einmal untersucht worden ist. »C« wird später die Summe aus »R«, »L«, »O« und »U«.

Die nächsten vier Zeilen sehen recht kompliziert aus und enthalten die oben genannten vier Variablen. Gehen wir die Zeilen langsam durch. »RX« und »RY« enthalten Zufallswerte zwischen eins und 19 bzw. eins und elf. Am Anfang der folgenden Zeilen steht immer eine Abfrage. Dadurch wird geprüft, ob »RX« oder »RY« am Spielfeldrand stehen. Das ist

erfüllt, wenn »RX« und »RY« jeweils den größten oder kleinsten Wert (eins, elf oder 19) annehmen. Sollte das der Fall sein, kann das Feld in Richtung Rand nicht mehr überprüft werden.

Beispiel: »RX« steht auf 19, das heißt, das überprüfte Feld befindet sich irgendwo am rechten Feldrand. Jetzt soll mit Hilfe von »RX+1« (siehe Zeile 1210) die Nachbarzelle überprüft werden. »RX+1« ergibt 20, also wäre es unsinnig, diese Koordinate einzusetzen; und gerade das soll die Abfrage ja vermeiden, die für »RX=19« bereits nicht mehr erfüllt ist!

Aber was wird jetzt eigentlich überprüft, bzw. was wird den einzelnen Variablen zugeordnet? Wie gesagt enthält »MA(RX+1, RY, 0)« den Wert des Nachbarfeldes. Jetzt muß ich Ihnen etwas über den Aufbau des Arrays »MA« erzählen:

MA(„0) enthält, wie oft dieses Feld bereits untersucht worden ist.
MA(„1) Ist der Weg in die rechte Nachbarzelle frei?
MA(„2) Ist der Weg in die untere Nachbarzelle frei?
MA(„3) Ist der Weg in die linke Nachbarzelle frei?
MA(„4) Ist der Weg in die obere Nachbarzelle frei?

Das gilt für alle »RX«- und »RY«-Werte! Mit dem Teil »MA(RX+1,RY,0)>0« wird getestet, ob dieses Feld bereits »betreten« worden ist. Für diesen Test gibt es zwei mögliche Ergebnisse: -1 für wahr und 0 für falsch (Sie erinnern sich?). Dieses Ergebnis wird noch mit -1 multipliziert (indem man einfach ein Minuszeichen vor die Klammern setzt). Als Endergebnis kann für »R« demnach null (falls das Feld noch nicht betreten wurde) oder eins (wenn das Feld bereits ein oder mehrere Male untersucht wurde) herauskommen (-0 ist mathematisch unsinnig).

Diese Ausführungen gelten natürlich analog für die nächsten drei Zeilen. Nur werden da die Nachbarzellen in den drei anderen Richtungen überprüft und die Werte den anderen Variablen zugewiesen.

Abschließend werden in »C« die vier »Richtungsvariablen« summiert. Sollte C=4 sein (Zeile 1260), heißt das, daß alle vier angrenzenden Nachbarzellen schon einmal betreten wurden. Ist diese Bedingung erfüllt oder »C« gleich drei bzw. vier (C>2) *und* eine Zufallszahl zwischen eins und zehn kleiner als drei, wird nach 1720 verzweigt. In diesen Fällen kann man mit der gerade untersuchten Zelle bei »RX« und »RY« nicht mehr viel anfangen.

Wenn ein Feld nämlich schon einmal betreten wurde, sollte man die Mauer zu diesem Feld nicht unbedingt durchbrechen (sonst hat man am Schluß ein Labyrinth ohne Mauern!). Daher diese Abfrage, ob das Nachbarfeld schon einmal untersucht worden ist.

Sind aber mindestens zwei Nachbarzellen noch frei, kommt der Computer zu Zeile 1270. Dort verzweigt er - je nach Zufallszahl nach 1300, 1400, 1500 oder 1600, wo die Variablen »R«, »L«, »O« und »U« untersucht werden, um eine Mauer zu durchbrechen.

Anhand der Zeilen 1300 bis 1380 möchte ich den Aufbau und die Funktionsweise erklären, um dann bei den restlichen drei Routinen nur noch auf die Unterschiede eingehen zu

müssen. Zunächst wird in Zeile 1300 überprüft, ob die Mauer zu dem Feld auf der rechten Seite überhaupt durchbrochen werden soll. Sie müssen jetzt die beiden Felder gut auseinanderhalten. Im Moment wird die »Zelle« RX/RY untersucht. Von ihr aus soll (wenn R<>0 ist) nach rechts eine Verbindung hergestellt werden. Das bedeutet, daß von dem Feld »RX+1/RY« eine Verbindung nach links geschaffen werden soll. Und das sehen wir uns jetzt mal im Programmtext an:

Ist R<>0 (ungleich null), dann kann das Programm zur nächsten Überprüfung weiterspringen, da in diesem Fall nichts passieren soll. Ist »R« dagegen null (die Bedingung in Zeile 1300 ist dann nicht erfüllt), darf man Hammer und Meißel zur Hand nehmen ...

Zunächst wird der Bildschirm an die Stelle der jetzigen Zelle gesetzt. Die genauen Cursor-Daten sind wieder einmal durch Herumprobieren entstanden.

An dieser Stelle wird ein Mauerteil gelöscht (Zeile 1320). Daraufhin muß »MA(RX,RY,0)« um eins erhöht werden, da diese Stelle im Moment »begangen« wird. Daraufhin wird festgesetzt, daß der Weg nach rechts frei ist (MA(RX,RY,1)=1). Eins steht an dieser Stelle für frei, null für belegt. Somit kann man die Abfragen wieder ganz einfach gestalten.

Anschließend wird »RX« um eins erhöht. Zum einen, da der Inhalt der Nachbarzelle noch aktualisiert werden muß, und zum anderen, weil das Programm versucht, immer mehrere Mauerdurchbrüche hintereinander zu schaffen. Dadurch werden längere Gänge »garantiert«. Das Programm arbeitet auch schneller, als wenn es nach jedem einzelnen Durchbruch sich eine neue Stelle suchen müßte.

Nachdem »RX« erhöht wurde, kann man es für die Nachbarzelle verwenden; und bei der wird festgesetzt, daß ab sofort der Weg nach links frei ist (MA(RX,RY,3)=1). Zu guter Letzt muß aber noch gespeichert werden, daß auch diese Zelle einmal mehr betreten wurde (Zeile 1370). Abschließend springt das Programm nach Zeile 1700; dort steht eine Überprüfung, doch dazu später mehr.

Die anderen drei Routinen arbeiten nach demselben Prinzip. Nur muß man dort jeweils eine andere Variable (»L«,»O« oder »U« anstelle von »R«) untersuchen, »RX/RY« dementsprechend ändern, und auch bei beiden Zellen die korrekten Richtungen belegen. Aber ich glaube, das dürfte jetzt nicht mehr so schwer sein. Probieren Sie es einmal. Beißen Sie sich durch!

Ich setze jetzt wieder bei Zeile 1700 ein. Dort werden im Laufe der Vorbereitungen alle Zellen erfaßt, die genau einmal beschritten worden sind (als Zählvariable wird »CN« benutzt). Da eine Zelle nicht bei einem Durchgang gleich zweimal untersucht werden kann, werden nach und nach alle Stellen erfaßt. Ich meine damit, daß eine Zelle erfaßt wurde, auch wenn sie mittlerweile vielleicht für MA(RX,RY,0) den Wert fünf hat. Beim ersten Mal wurde sie mitgezählt.

Insgesamt verfügt das Labyrinth über 11*19=209 Stellen. Und erst, wenn jede einzelne Stelle betreten worden ist (sprich in 1700 erfaßt wurde) und »CN« den Wert 209 angenommen hat, wird dieser Programmteil durch einen Sprung nach 1800 verlassen.

Andernfalls springt das Programm wieder an den Anfang der Überprüfungen. Das heißt natürlich nicht, daß jetzt wieder ganz von vorne angefangen wird. Aber das Programm setzt seine Untersuchungen fort.

Von Zeile 1260 aus wird nach Zeile 1720 verzweigt. Das heißt, die Zeilen 1720 bis 1750 werden nur dann bearbeitet, wenn die Bedingung in 1260 erfüllt ist (siehe dort). In dem Fall müssen »RX« und »RY« neu bestimmt werden. Sollte das Feld an der Stelle RX/RY schon einmal betreten worden sein, laufen die Tests sofort weiter (1730). Andernfalls sucht der C64 weiter, und zwar so lange, bis er ein bereits untersuchtes Feld vorfindet (siehe Zeilen 1740 und 1750). Durch diese Bedingung soll verhindert werden, daß Gänge erstellt werden, die keinerlei Verbindung zur Außenwelt haben. Es wäre wenig sinnvoll, vielleicht drei separate Gänge innerhalb des Labyrinths zu haben. Sie hätten keine Chance, jemals an den Ausgang zu kommen (haben Sie so auch nicht, aber das macht ja nichts!).

Ab Zeile 1800 entsteht der endgültige Bildschirmaufbau. Dazu wird einfach ein Rahmen um den Bildschirm gezogen (1800 bis 1870). In 1880 werden Ihre Startkoordinaten festgelegt (PX/PY wie Player). Sie sollen also auf alle Fälle in der ersten Spalte beginnen. Die Y-Koordinate kann jedoch zwischen eins und elf (der gesamten Labyrinthbreite) schwanken. Daraufhin wird Ihr Zeichen (ein kleiner Ball) an der entsprechenden Stelle ausgegeben. Vielleicht wundern Sie sich schon über das PX+PX bzw. PY+PY? Wenn Sie sich die Ausmaße des Labyrinths in 1030 anschauen und mit der Bildschirmdarstellung vergleichen, werden Sie wissen, woher das kommt. Da zwischen zwei freien Feldern (ursprünglich) eine Mauer war, kann Ihr Zeichen nur an jeder zweiten Stelle auf dem Bildschirm sitzen. Und das erreicht man mit 2*PX=PX+PX. Die »-1« sind immer eine Korrektur, um wirklich an die richtige Stelle zu kommen. Sie können das ja mal weglassen und sehen, was dann passiert ...

In den Zeilen 1910 bis 1930 wird der Ausgang festgelegt. Er muß an der X-Koordinate 38 (siehe 1920) liegen. Das entspricht in unserem Array »MA« dem Wert 19. Lediglich die Höhe des Ausgangs kann – wie Sie selbst – zwischen eins und elf schwanken. Den Ausgang können Sie am rechten Bildschirmrand als schwarzes Quadrat erkennen.

In den Zeilen 1940 bis 1960 wird zum Schluß noch der Drache festgelegt. Er muß die X-Koordinate 37 haben, sprich, er sitzt eine Spalte vor dem Ausgang. Und zwar direkt daneben, wie MY=WY beweist; er hat nämlich dieselbe Y-Koordinate wie der Ausgang.

In den Zeilen 1970 bis 2010 wird noch die Variable MA(,,0) zurückgesetzt. Denn jetzt braucht man den Inhalt nicht mehr. Aber die Variable wird später wieder benötigt. Zu guter Letzt erhalten Sie noch eine Kurzanleitung auf dem Bildschirm, die Ihnen lediglich angibt, mit welchen Tasten Sie Ihr Zeichen auf dem Bildschirm hin- und herbewegen können.

Der Programmteil »SPIEL« geht »medias in res« (in die Vollen, etwas frei übersetzt). Er beginnt nämlich gleich mit der TAstaturabfrage (TastAturabfrage, TastaturAbfrage oder gar TastaturabfrAge?!), in dem er der Variablen »TA« den Inhalt des Tastaturpuffers übergibt. Wurde noch keine Taste gedrückt, kehrt das Programm gleich wieder nach 2130 zurück. Ansonsten springt es entsprechend den Buchstaben zu den einzelnen Routinen.

Wenn Sie dagegen eine nicht erlaubte Taste gedrückt haben, haben Sie Ihre Chance verspielt, und der Drache kommt zum Zug (Zeile 2200).

Auch hier möchte ich wieder die Funktionsweise an einem Beispiel erklären, damit Sie die anderen drei Routinen selbst »nachspielen« können.

Als erstes wird immer überprüft, ob der Weg in der angegebenen Richtung frei ist. Dazu werden die Elemente des Arrays »MA« benutzt (siehe jeweils letzten Index in den Zeilen 2250, 2300, 2350 und 2400). Sollten sich keine Probleme ergeben, werden die Hilfsvariablen »X2« und »Y2« jeweils mit den neuen Koordinaten belegt, und es erfolgt ein Sprung nach 2450.

War jedoch eine Mauer im Weg, wird diese in den nächsten drei Zeilen gezeichnet. Dazu benutzt das Programm eine Schleife, die dreimal durchlaufen wird. Sehen Sie selbst, wie elegant man mit der Laufvariablen die Y-Koordinate in Zeile 2270 korrigieren kann!

Abschließend erfolgt ein Sprung nach Zeile 2500, um den Drachen zu bewegen. Die folgenden drei Programmteile arbeiten nach demselben Schema. Dabei sind die Zeilen 2300 bis 2340 für die Richtung »nach unten«, 2350 bis 2390 »nach links« und 2400 bis 2440 »nach oben« zuständig.

Nach 2450 wurde immer verzweigt, wenn der Weg frei war; X2 und Y2 enthalten hier die neuen Koordinaten. Zunächst wird der Cursor auf die alten Werte »PX/PY« gesetzt, um Ihr Zeichen zu löschen. Wenn Sie diese Zeile entfernen (ich meine nur 2460), können Sie immer verfolgen, wo Sie schon überall waren. 2450 dürfen Sie nicht löschen, ohne vorher die Sprünge in 2250, 2300, 2350 und 2400 angepaßt zu haben; sonst kommt es zu einer Fehlermeldung!

Nachdem die alte Darstellung gelöscht wurde, werden die Koordinaten »PX/PY« aktualisiert, sowie Ihr Zeichen an der neuen Stelle ausgegeben (2480 und 2490).

In Zeile 2500 wird zunächst geprüft, ob Sie nicht zufällig gerade den Ausgang erreicht haben. Stimmen beide Koordinaten überein, wird die Variable »WI« auf »−1« gesetzt. Diese Variable wird ab Zeile 3000 noch Bedeutung gewinnen.

In Zeile 2510 wird geprüft, ob Ihre Koordinaten aber nicht vielleicht mit denen des Drachens zusammenfallen. Wenn das nicht der Fall ist, kann das Programm nach 2560 verzweigen; andernfalls würde der Drachen über Ihr Zeichen gesetzt (2520 und 2530), »WI« auf null gesetzt und mit »RETURN« zurückgekehrt.

Die Zeilen ab 2560 werden folglich bearbeitet, wenn Sie weder am Ausgang sind noch vom Drachen gefressen werden. Mittels einer Zufallszahl wird bestimmt, welche Koordinaten geprüft werden sollen. Dadurch erhalten Sie eine Chance, da der Drachen nicht immer alle Koordinaten berücksichtigt, sondern − je nach Zufallszahl − nur die Abfragen ab 2580, 2590 oder 2600 abarbeitet.

Die folgenden Routinen werden immer nur angesprungen, falls die dazugehörige Abfrage in 2570 bis 2600 überhaupt erreicht wurde. Unter dieser Voraussetzung behandle ich die folgenden Zeilen; ich möchte nicht immer »unter der Bedingung ...« dazusagen müssen.

Die Zeilen 2620 bis 2650 werden angesprungen, wenn das Programm festgestellt hat, daß sich der Drache links von Ihnen befindet, die Zeilen 2660 bis 2690, falls er sich oberhalb von Ihnen, 2700 bis 2730, wenn er sich rechts von Ihnen und 2740 bis 2770 schließlich, wenn er sich unterhalb von Ihnen befindet. Die Feinheiten überlasse ich Ihnen. Viel Erfolg!

Sobald der Interpreter in Zeile 2800 angekommen ist, stehen in »X2« und »Y2« die neuen Koordinaten. Mit den Zeilen 2800 bis 2840 wird die alte Darstellung des Drachen gelöscht und sein Zeichen an der neuen Position gesetzt (analog zu den Zeilen 2450 bis 2490). In 2860 und 2870 wird noch einmal getestet, ob Sie vielleicht den Ausgang erreicht haben, bzw. ob der Drache Sie mittlerweile eingeholt hat. Falls dem so ist, wird dementsprechend die Variable »WI« gesetzt. Und welche Bedeutung diese Variable hat, können Sie anhand der Zeilen 3030 bis 3070 sehen.

17 Minigolf

Einführung

Dieses Programm ist eines der längsten in diesem Buch, was aber nicht gleichbedeutend mit einem der schwersten ist. Ein Großteil des Spiels besteht nur aus Anweisungen für den Bildschirmaufbau. Aber zu diesem Thema werden wir erst in der genauen Programmbeschreibung kommen.

Wenn Sie das Programm starten, sehen Sie gleich den ersten »Parcours« vor sich. Wenn Sie sich die Bahn so betrachten, werden Sie wohl mit mir einer Meinung sein, daß man dieses Spiel weder als Minigolf noch als Golf bezeichnen kann. Das »Spielfeld« deutet eher auf Golf hin, aber andererseits ist dieser Platz wieder so mini ...

Am oberen Bildschirmrand sehen Sie einige Informationen. Zum einen wird da angezeigt, auf welcher Bahn Sie sich gerade befinden (»LOCH #«). Desweiteren können Sie erkennen, wie viele Schläge Sie in etwa benötigen sollten (»PAR:«), und wie viele Schläge Sie bereits gebraucht haben.

Darunter werden mögliche Hindernisse wie Wasser, Sand, Unebenheiten und Blöcke angezeigt. Darauf sollten Sie sich also gefaßt machen.

Gleich darunter erscheint eine Minianleitung. Daran können Sie erkennen, welche Zahl Sie für welche Richtung eingeben sollen:

2 1 8
3 * 7
4 5 6

Es dürfte klar sein, daß Sie meinetwegen [1] eingeben müssen, um den Ball nach oben zu schlagen, [2] für schräg oben (mehr nach links), während [8] für schräg nach oben rechts zuständig ist ...

Auf der Bahn können Sie Ihren Ball und das Loch (schwarzes Quadrat) erkennen. Irgendwie sollten Sie den Ball mit den möglichen Richtungen zum Loch bringen.

Dazu erscheint unterhalb des Spielfeldes eine Abfrage, die zum einen wissen will, in welche Richtung Sie schlagen wollen, und zum anderen, mit welcher Kraft der Schlag ausgeführt werden soll. Ich verrate hier nicht, wie weit Sie mit welcher Kraft kommen (das werden Sie später noch sehen). Aber eines ist klar: Sobald Sie Ihren Ball in Sand oder Wasser (kleine Pfützen sind hier wohl gemeint) setzen, benötigen Sie sehr viel mehr Kraft als auf dem Gras. Aber ansonsten dürfen Sie ruhig mal probieren.

Übrigens wird der Ball an der Spielfeldbegrenzung reflektiert. Er verschwindet also nicht auf Nimmerwiedersehen. Manchmal ist es ganz lustig, mit der maximalen Stärke gegen die Begrenzung zu spielen (»mit Bande« im Billarddeutsch).

Zur Schlagstärke möchte ich aber noch etwas sagen: Die Werte müssen zwischen null und fünf liegen; das heißt aber nicht, daß die Werte ganzzahlig sein müssen. Das Programm verarbeitet genausogut Eingaben wie »1.2«, »4.9« etc.

Nur bei der Richtung sollten Sie darauf achten, ganze Zahlen einzugeben. Ihnen stehen bereits acht Richtungen zur Verfügung, so daß eine weitere Unterteilung nicht mehr besonders sinnvoll erscheint. Vor allem würde der Programmieraufwand »ins Unermeßliche steigen«. Das nicht gerade, aber mit acht Richtungen ist doch auch schon was anzufangen, oder?

Und jetzt viel Spaß, wenn Sie alle Rekorde brechen!

Listing zu Minigolf

```
10 REM ****************
20 REM ***          ***
30 REM *** MINIGOLF ***
40 REM ***          ***
50 REM ****************
60 REM
70 GOSUB 1000
80 GOSUB 2500
90 GOSUB 3700
100 END
1000 REM
1010 REM *** VORBEREITUNGEN ***
1020 REM
1030 DIM HA(5,5)
1040 FOR I=1 TO 5
1050 :FOR J=1 TO 5
1060 : READ HA(I,J)
1070 :NEXT J
1080 NEXT I
1090 DATA 0,0,0,0,2
1100 DATA 1,0,0,0,3
1110 DATA 0,1,0,0,3
1120 DATA 1,1,0,0,3
1130 DATA 1,0,1,0,3
1180 DIM RI(8,2)
1190 FOR I=1 TO 8
1200 :READ RI(I,1), RI(I,2)
1210 NEXT I
1220 DATA 0,-1,-1,-1,-1,0,-1,1
1230 DATA 0,1,1,1,1,0,1,-1
1240 DIM V(8)
1250 FOR I=1 TO 8
1260 :READ V(I)
1270 NEXT I
1280 DATA -40,-41,-1,39,40,41,1,-39
1300 DEF FN FA(X)=PEEK(55296+BX+40*BY+X) AND 15
1310 DEF FN BS(X)=PEEK(1024+BX+40*BY+X)
1320 RETURN
1350 REM ***************** PLATZ 1
1360 GOSUB 1300
1370 PRINT "<HOME>"
1380 PRINT TAB(10);"<BLACK>DTTTTTTTTTTTTTTTTTTF"
1390 FOR I=1 TO 15
1400 :PRINT TAB(10);"<RVSON>K<GREEN,18SPACE,RVOFF,BLACK>K"
1410 NEXT I
1420 PRINT TAB(10);"C<RVSON>TTTTTTTTTTTTTTTTTT<RVOFF>V"
1430 PRINT "<HOME,4DOWN>";TAB(19);"<RVSON,SPACE>"
1440 BX=INT (RND(1)*17)+11:BY=16
1450 POKE 211,BX:POKE 214,BY:SYS 58640
1460 PRINT "<RVSON,GREEN>Q";
1470 RETURN
1480 REM ***************** PLATZ 2
1490 GOSUB 1300
1500 PRINT "<HOME>"
1510 PRINT TAB(8);"<BLACK>DTTTTTTTTTTTTTTTTTTTTTTTTF"
1520 FOR I=1 TO 5
1530 :PRINT TAB(8);"<RVSON>K<GREEN,24SPACE,RVOFF,BLACK>K"
1540 NEXT I
1550 PRINT TAB(8);"<RVSON>K<GREEN,9SPACE,BLUE,3SPACE,BLACK,SPACE>TTTTTTT
     TTTT<RVOFF>V"
1560 PRINT TAB(8);"<RVSON>K<GREEN,8SPACE,BLUE,4SPACE,RVOFF,BLACK>K"
1570 PRINT TAB(8);"<RVSON>K<GREEN,7SPACE,BLUE,5SPACE,RVOFF,BLACK>K"
```

```
1580 PRINT TAB(8);"{RVSON}T{GREEN,8SPACE,BLUE,4SPACE,BLACK,SPACE,RVOFF}T"
1590 PRINT TAB(8);"{RVSON}T{GREEN,13SPACE,RVOFF,BLACK}T"
1600 FOR I=1 TO 6
1610 :PRINT TAB(8);"{RVSON}T{GREEN,13SPACE,RVOFF,BLACK}T"
1620 NEXT I
1630 PRINT TAB(8);"C{RVSON}TTTTTTTTTTTTT{RVOFF}V"
1640 BX=INT (RND(1)*13)+9:BY=17
1650 POKE 211,BX:POKE 214,BY:SYS 58640
1660 PRINT "{RVSON,GREEN}Q"
1670 PRINT "{HOME,4DOWN}";TAB(30);"{RVSON,BLACK,SPACE}"
1680 RETURN
1690 REM ***************** PLATZ 3
1700 PRINT"{HOME}"
1710 PRINT TAB(20);"{BLACK}DTTTTTTTTTTTTTF"
1720 FOR I=1 TO 5
1730 :PRINT TAB(20);"{RVSON}T{GREEN,13SPACE,RVOFF,BLACK}T"
1740 NEXT I
1750 PRINT TAB(8);"{RVOFF}DTTTTTTTTTTT{RVSON}V{GREEN,13SPACE,RVOFF,BLACK}T"
1760 PRINT TAB(8);"{RVSON}T{GREEN,22SPACE,RVOFF,BLACK}T"
1770 PRINT TAB(8);"{RVSON}T{GREEN,12SPACE,YELLOW,10SPACE,RVOFF,BLACK}T"
1780 PRINT TAB(8);"{RVSON}T{GREEN,11SPACE,YELLOW,11SPACE,RVOFF,BLACK}T"
1790 PRINT TAB(8);"{RVSON}T{GREEN,11SPACE,YELLOW,11SPACE,RVOFF,BLACK}T"
1800 PRINT TAB(8);"{RVSON}T{GREEN,12SPACE,YELLOW,4SPACE,GREEN,6SPACE,RVO
     FF,BLACK}T"
1810 PRINT TAB(8);"{RVSON}T{GREEN,13SPACE,BLACK}TTTTTTTTT{RVOFF}V"
1820 FOR I=1 TO 4
1830 :PRINT TAB(8)"{RVSON}T{GREEN,13SPACE,RVOFF,BLACK}T"
1840 NEXT I
1850 PRINT TAB(8);"{RVOFF}C{RVSON}TTTTTTTTTTTTTT{RVOFF}V"
1860 PRINT "{HOME,3DOWN}";TAB(32);"{RVSON,BLACK,SPACE}"
1870 BX=INT (RND(1)*13)+9:BY=17
1880 POKE 211,BX:POKE 214,BY:SYS 58640
1890 PRINT "{GREEN,RVSON}Q"
1900 RETURN
1910 REM ***************** PLATZ 4
1920 PRINT "{HOME}"
1930 PRINT TAB(7);"{BLACK}DTTTTTTTTTTTTTTTTTTTTTTTTTTTF"
1940 FOR I=1 TO 6
1950 :PRINT TAB(7);"{RVSON}T{GREEN,27SPACE,RVOFF,BLACK}T"
1960 NEXT I
1970 PRINT TAB(7);"{RVSON}T{GREEN,6SPACE,BLACK}DTTTTTF{GREEN,14SPACE,RVO
     FF,BLACK}T"
1980 FOR I=1 TO 4
1990 :PRINT TAB(7);"{RVSON}T{GREEN,6SPACE,BLACK,RVOFF}T{5SPACE,RVSON}T{G
     REEN,14SPACE,RVOFF,BLACK}T"
2000 NEXT I
2010 PRINT TAB(7);"{RVSON}T{GREEN,6SPACE,BLACK,RVOFF}T{5SPACE}C{RVSON}TT
     TTTTTTTTTTTT{RVOFF}V"
2100 FOR I=1 TO 3
2110 PRINT TAB(7);"{RVSON}T{GREEN,6SPACE,BLACK,RVOFF}T"
2120 NEXT I
2130 PRINT TAB(7);"C{RVSON}TTTTTT{RVOFF}V"
2140 PRINT "{HOME,2DOWN}";TAB(18);"{YELLOW,RVSON,7SPACE,DOWN,6LEFT,6SPAC
     E,DOWN,4LEFT,SPACE}"
2150 PRINT TAB(25);"{BLUE,RVSON,SPACE,DOWN,3LEFT,3SPACE,DOWN,5LEFT,6SPACE}"
2160 POKE 211,32:POKE 214,10:SYS 58640
2170 PRINT "{RVSON,BLACK,SPACE}"
2180 BX=INT (RND(1)*5)+9:BY=16
2190 POKE 211,BX:POKE 214,BY:SYS 58640
2200 PRINT "{RVSON,GREEN}Q"
2210 RETURN
2220 REM *********************** PLATZ 5
2230 PRINT "{HOME}"
```

```
2240 PRINT TAB(8);"{BLACK}DTTTTTTTTTTTTF"
2250 FOR I=1 TO 4
2260 :PRINT TAB(8);"{RVSON}K{GREEN,12SPACE,RVOFF,BLACK}K"
2270 NEXT I
2280 PRINT TAB(8);"{RVSON}K{GREEN,12SPACE,BLACK}C{RVOFF}TTTTTTTTTTTTTTTTTF"
2290 FOR I=1 TO 4
2300 :PRINT TAB(8);"{RVSON}K{GREEN,29SPACE,BLACK,RVOFF}K"
2310 NEXT I
2320 PRINT TAB(8);"{BLACK}C{RVSON}TTTTTTTTTTTTTTTF{GREEN,13SPACE,BLACK,R
     VOFF}K"
2330 FOR I=1 TO 5
2340 :PRINT TAB(24);"{RVSON,BLACK}K{GREEN,13SPACE,BLACK,RVOFF}K"
2350 NEXT I
2360 PRINT TAB(24);"{BLACK}C{RVSON}TTTTTTTTTTTTT{RVOFF}V"
2370 POKE 211,9:POKE 214,7:SYS 58640
2380 PRINT "{BLUE,RVSON,4SPACE,DOWN,4LEFT,4SPACE,DOWN,4LEFT,5SPACE,DOWN,
     5LEFT,6SPACE}"
2390 POKE 211,27:POKE 214,7:SYS 58640
2400 PRINT "{RED,RVSON,7SPACE,DOWN,6LEFT,6SPACE,DOWN,6LEFT,7SPACE,DOWN,7
     LEFT,5SPACE}"
2410 POKE 211,13:POKE 214,4:SYS 58640
2420 PRINT "{RVSON,BLACK,SPACE}"
2430 BX=INT (RND(1)*13)+25:BY=16
2440 POKE 211,BX:POKE 214,BY:SYS 58640
2450 PRINT "{RVSON,GREEN}Q"
2460 RETURN
2500 REM
2510 REM *** SPIEL ***
2520 REM
2530 FOR HO=1 TO 5
2540 :BC$="{GREEN}"
2550 :PRINT "{CLR}";
2560 :ON HO GOSUB 1350, 1480, 1690, 1910, 2220
2570 :PRINT "{HOME,RVSON,LIG.BLUE}LOCH #";HO;"PAR:";HA(HO,5);"SCHLAEGE:";SC
2580 :PRINT:PRINT "HINDERNIS:"
2590 :IF HA(HO,1) THEN PRINT "WASSER"
2600 :IF HA(HO,2) THEN PRINT "SAND"
2610 :IF HA(HO,3) THEN PRINT "UNEBEN"
2620 :IF HA(HO,4) THEN PRINT "BLOECKE"
2630 :PRINT:PRINT "{RVSON,BLACK}2 1 8"
2640 :PRINT "{RVSON}3 Q 7"
2650 :PRINT "{RVSON}4 5 6"
2660 :POKE 211,0:POKE 214,19:SYS 58640
2670 :FOR I=1 TO 5
2680 : PRINT "{39SPACE}"
2690 :NEXT I
2700 :POKE 211,0:POKE 214,20:SYS 58640
2710 :D$="":INPUT "RICHTUNG (1-8) ";D$
2720 :RI=VAL(D$)
2730 :IF RI>=1 AND RI<=8 THEN GOTO 2760
2740 :PRINT "BITTE NUR 1, 2, 3,..., 8 EINGEBEN !"
2750 :GOTO 2700
2760 :POKE 211,0:POKE 214,19:SYS 58640
2770 :FOR I=1 TO 5
2780 : PRINT "{39SPACE}"
2790 :NEXT I
2800 :POKE 211,0:POKE 214,20:SYS 58640
2810 :D$="":INPUT "GESCHWINDIGKEIT (0-5) ";D$
2820 :GS=VAL(D$)
2830 :IF GS>=0 AND GS<=5 THEN GOTO 2860
2840 :PRINT "BITTE NUR 0 BIS 5 EINGEBEN !"
2850 :GOTO 2800
2860 :UF=0
2870 :TF=0
```

```
2880 :X2=BX+RI(RI,1):Y2=BY+RI(RI,2)
2890 :D2=V(RI)
2900 :FA=FN FA(D2):IF FA<>5 OR FN BS(D2)<>160 THEN GOTO 2980
2910 :POKE 211,BX:POKE 214,BY:SYS 58640
2920 :PRINT BC$;"{RVSON,SPACE}";
2930 :BX=X2:BY=Y2
2940 :POKE 211,BX:POKE 214,BY:SYS 58640
2950 :PRINT "{RVSON,GREEN}Q";
2960 :BC$="{GREEN}"
2970 :GOTO 3640
2980 :IF FA=0 AND FN BS(D2)<>160 THEN GOTO 3100
2990 :IF FA=6 THEN GOTO 3290
3000 :IF FA=7 THEN GOTO 3381
3010 :IF FA=2 THEN GOTO 3400
3020 :IF FA=0 THEN GOTO 3520
3100 :IF RI=1 OR RI=3 THEN RI=RI+4:GOTO 2880
3110 :IF RI=5 OR RI=7 THEN RI=RI-4:GOTO 2880
3120 :ON RI/2 GOTO 3130, 3170, 3210, 3250
3130 :IF FN BS(D2+1)<>160 AND FN BS(D2+40)<>160 THEN RI=6:GOTO 2880
3140 :IF FN BS(D2+1)<>160 THEN RI=4:GOTO 2880
3150 :IF FN BS(D2+40)<>160 THEN RI=8:GOTO 2880
3160 :RI=6:GOTO 2880
3170 :IF FN BS(D2+1)<>160 AND FN BS(D2-40)<>160 THEN RI=8:GOTO 2880
3180 :IF FN BS(D2+1)<>160 THEN RI=2:GOTO 2880
3190 :IF FN BS(D2-40)<>160 THEN RI=6:GOTO 2880
3200 :RI=8:GOTO 2880
3210 :IF FN BS(D2-1)<>160 AND FN BS(D2-40)<>160 THEN RI=2:GOTO 2880
3220 :IF FN BS(D2-1)<>160 THEN RI=8:GOTO 2880
3230 :IF FN BS(D2-40)<>160 THEN RI=4:GOTO 2880
3240 :RI=2:GOTO 2880
3250 :IF FN BS(D2-1)<>160 AND FN BS(D2+40)<>160 THEN RI=4:GOTO 2880
3260 :IF FN BS(D2-1)<>160 THEN RI=6:GOTO 2880
3270 :IF FN BS(D2+40)<>160 THEN RI=2:GOTO 2880
3280 :RI=4:GOTO 2880
3290 :IF TF THEN GOTO 3310
3300 :TF=3:SC=SC+1
3310 :POKE 211,BX:POKE 214,BY:SYS 58640
3320 :PRINT BC$;"{RVSON,SPACE}";
3340 :BX=X2:BY=Y2
3350 :POKE 211,BX:POKE 214,BY:SYS 58640
3360 :PRINT "{BLUE,RVSON}Q";
3370 :BC$="{BLUE}"
3380 :GOTO 3640
3381 :IF TF THEN GOTO 3383
3382 :TF=3:SC=SC+1
3383 :POKE 211,BX:POKE 214,BY:SYS 58640
3384 :PRINT BC$;"{RVSON,SPACE}";
3385 :BX=X2:BY=Y2
3386 :POKE 211,BX:POKE 214,BY:SYS 58640
3387 :PRINT "{YELLOW,RVSON}Q";
3388 :BC$="{YELLOW}"
3389 :GOTO 3640
3400 :IF UF THEN GOTO 3450
3410 :UF=1
3420 :RI=RI+INT (RND(1)*2)*2-1
3430 :IF RI=0 THEN RI=8
3440 :IF RI=9 THEN RI=1
3450 :POKE 211,BX:POKE 214,BY:SYS 58640
3460 :PRINT BC$;"{RVSON,SPACE}";
3470 :BX=X2:BY=Y2
3480 :POKE 211,BX:POKE 214,BY:SYS 58640
3490 :PRINT "{RED,RVSON}Q";
3500 :BC$="{RED}"
3510 :GOTO 3640
```

```
3520 :POKE 211,BX:POKE 214,BY:SYS 58640
3530 :PRINT "{RVSON,GREEN,SPACE}"
3540 :POKE 211,X2:POKE 214,Y2:SYS 58640
3550 :PRINT "{RVSON,WHITE,SPACE}";
3560 :PRINT "{LEFT,RVSON,BLACK,SPACE}";
3570 :RI=RI+INT (RND(1)*2)*2-1
3580 :IF RI=0 THEN RI=8
3590 :IF RI=9 THEN RI=1
3600 :GS=GS-0.5
3610 :IF GS>0 THEN X2=X2+RI(RI,1):Y2=Y2+RI(RI,2):GOTO 2890
3620 :SC=SC+1
3630 :GOTO 3680
3640 :IF TF>0 THEN TF=TF-1:IF TF=0 THEN GOTO 3660
3650 :GS=GS-0.3:IF GS>0 THEN GOTO 2880
3660 :SC=SC+1:GOTO 2570
3670 :GET X$:IF X$="" THEN GOTO 3670
3680 NEXT HO
3690 RETURN
3700 REM
3710 REM *** ENDE ***
3720 REM
3730 PRINT "{CLR,3DOWN,LIG.BLUE}"
3740 PRINT "DAS SPIEL IST BEENDET."
3750 PRINT "{DOWN}SIE HABEN ALLES IN ALLEM";SC
3760 PRINT "SCHLAEGE BENOETIGT."
3770 PRINT "DAS SIND{2SPACE}DURCHSCHNITTLICH";INT (SC/5*100)/100
3780 PRINT "SCHLAEGE PRO BAHN."
3790 PRINT "BIS ZUM NAECHSTEN MAL !"
3800 RETURN
```

■ Programmbeschreibung für Minigolf

So lang dieses Programm ist, so kurz dürfte die Erklärung dazu werden. Denn der größte Teil ist – wie bereits bei der Einführung erwähnt – für die Bildschirmdarstellung zuständig. Nichtsdestoweniger hat auch dieses Programm so seine Tücken, die erklärt sein wollen.

Zunächst wird ein zweidimensionales Feld deklariert. Es soll jeweils fünf Elemente pro Index enthalten, sprich 25 Elemente insgesamt (der Index null bleibt mal wieder unberücksichtigt). In der verschachtelten Schleife (1040 bis 1080) wird dieses Array mit Werten belegt, die das Programm aus den DATA-Zeilen ab 1090 liest. Diese Daten sind bereits so angeordnet, daß Ihnen die Zuordnung zu den einzelnen Variablen nicht allzuschwer fallen sollte.

Daraufhin wird ein weiteres zweidimensionales Feld eingerichtet, das für die Koordinatenänderung bei bestimmten Richtungen zuständig ist. Sie können dazu die DATA-Werte jeweils paarweise ordnen. Wenn Sie sich erinnern: Sie haben bei Minigolf acht mögliche Schlagrichtungen. Für jede einzelne Richtung geben diese Werte an, wie sehr die X- bzw. Y-Koordinate verändert werden sollen.

Beispiel: Nehmen wir an, Sie geben als Richtung [1] ein (für senkrecht nach oben). Dann soll die X-Koordinate gar nicht verändert, die Y-Koordinate jedoch um eins verringert werden. Und jetzt vergleichen Sie bitte das erste Zahlenpaar. Klar?

In Zeile 1240 wird ein eindimensionales Feld definiert, das acht Elemente hat. Auch dieses Array hat etwas mit der Richtung zu tun. Und zwar gibt es für jede Richtung an, wie viele Spalten man von der jetzigen Cursor-Position abziehen bzw. addieren muß, um an die neue Stelle in der entsprechenden Richtung zu kommen.

Beispiel: Angenommen, Sie wählen [7] für waagrecht nach rechts. Dann liegt die nächste Cursor-Position in dieser Richtung um eine Spalte daneben, man muß zur momentanen Bildschirmadresse eins addieren. Betrachten Sie sich bitte den siebten Wert in 1280: 1.

Haben Sie dagegen wieder [1] gewählt, muß man von der momentanen Bildschirmposition eine ganze Zeile (40 Spalten) abziehen, um an die neue Stelle zu kommen. Das erste Element ist »−40«. Dadurch, daß in dieser Liste negative und positive Zahlen verwendet werden, braucht man die aktuelle Bildschirmstelle und diese Veränderung nur zu addieren, da die Addition mit einer negativen Zahl dasselbe ist wie eine Subtraktion.

Abschließend werden noch zwei (bekannte) Funktionen definiert, die zu einer bestimmten Bildschirmstelle die FArbe bzw. den Inhalt (des BildSchirms) liefern.

Wenn Sie sich kurz noch diese beiden Funktionen betrachten: Als Argument wird ein Wert übergeben, der die Abweichung von der aktuellen Cursor-Position BX/BY darstellt. Wenn Sie beim Aufruf dieser Funktionen zum Beispiel irgendein V(I) einsetzen, kommen Sie genau an die gewünschte Bildschirmadresse!

Im folgenden werden fünf Plätze dargestellt. Wie schon früher, möchte ich nur einen näher erläutern, damit Sie sich selbst durch die anderen durcharbeiten können.

Zu Beginn jedes Platzes wird nach Zeile 1300 gesprungen, um sicher zu gehen, daß die Funktion mit den neuesten Werten ausgestattet ist. Eigentlich dürfte es nichts ausmachen, wenn Sie diese Sprünge weglassen ... Aber das können Sie einmal ausprobieren. Sollten sich Unregelmäßigkeiten ergeben, dann wissen Sie ja, woran das liegt!

In Zeile 1370 wird der Cursor an die Home-Position gesetzt (links oben im Bildschirm). Anschließend erfolgt die Ausgabe des Golfkurses. Das Einzige, was ich dazu erklären kann, sind die verwendeten Grafikzeichen. Und zwar wurden folgende Tastenkombinationen verwendet:

[SHIFT]+[I], [SHIFT]+[D], [SHIFT]+[F], [SHIFT]+[K], [CBM]+[C] und [CBM]+[V].

Hier wurden die Tastenkombinationen für Invers-an/aus und die einzelnen Farben nicht berücksichtigt. Es kommen aber lediglich die Farben Schwarz, Rot, Grün, Blau und Gelb vor, die Sie alle mit der [CTRL]-Taste erreichen können. Mit einem minimalen englischen Wortschatz sollten Sie diese Farben den Ziffertasten von eins bis acht zuordnen können.

Zeile 1430 setzt den Cursor an die Stelle 19/4. Dort wird das Loch ausgegeben (mit einem schwarzen Quadrat). Zeile 1440 bestimmt noch die Position Ihres Balles, der auch gleich ausgegeben wird (1450 und 1460). Ein »RETURN« schließt die Routine für Platz 1 ab.

Ich steige jetzt wieder bei Zeile 2530 ein. Da beginnt eine Schleife, die alle fünf Plätze »durchgeht«. In »BC$« wird die Farbe grün gespeichert. Diese Variable soll später dafür

sorgen, daß die Stelle, von der der Golfball weggeschossen wurde, nachher wieder so aussieht wie vorher. Dieses Prinzip ist Ihnen bereits aus vorhergehenden Spielen bekannt.

In Zeile 2550 wird der Bildschirm gelöscht. Wie Sie wissen, löschen die Routinen für die einzelnen Plätze den Bildschirm nicht, sondern setzen den Cursor lediglich in die linke obere Ecke. Daher muß der Bildschirm vor jeder Ausgabe einmal gelöscht werden.

In Zeile 2560 erfolgt ein Sprung zum gerade benötigten Golfplatz. Sobald der Computer also in 2570 ankommt, steht auf dem Bildschirm bereits der fertige Platz. Ab 2570 werden ein paar Informationen auf dem Bildschirm ausgegeben, damit der Benutzer weiß, woran er ist. Die Ausgaben brauche ich Ihnen nicht zu erklären. Aber betrachten Sie die Zeilen 2590 bis 2620. Hier sehen Sie, wie die Variable »HA« eingesetzt wird. Ich will das nicht ausführlich erklären, denn Sie müssen diese Zeilen eigentlich mit links beherrschen. Aber dennoch möchte ich auf die DATA-Zeilen aufmerksam machen. Da werden nämlich die Hindernisse wie Wasser etc. gespeichert. Sie sollten eigentlich in der Lage sein, diese paar Zeilen selbständig zu analysieren und *vor allem* zu verstehen!

Daß in den Zeilen 2660 bis 2690 die untersten fünf Zeilen gelöscht werden, dürfte auch nicht allzu schwer gewesen sein, oder? In den Zeilen 2700 bis 2850 erfolgt Ihre Eingabe. Sie werden nach der Richtung sowie der Schlagstärke gefragt. Zwischendurch (2760 bis 2790) werden noch einmal die untersten fünf Bildschirmzeilen gelöscht.

In den Zeilen 2860 und 2870 werden erst einmal zwei Flags gelöscht. Flags (auch Schalter genannt) zeigen immer einen Zustand an. Sie unterscheiden eigentlich nur die beiden Zustände, gesetzt und nicht gesetzt. Für unseren Fall heißt das, daß UF (wie Uneben-Flag) gesetzt ist, sobald es Unebenheiten gibt, bzw. UF nicht gesetzt ist, wenn das Feld »brettleben« ist. Das gleiche gilt für TF, das Hindernisse signalisiert.

In »BX« und »BY« sind die Koordinaten Ihres Golfballes gespeichert. Wie gesagt, enthält das Array RI die Veränderungen der beiden Koordinaten für jede Richtung. In Zeile 2880 werden den beiden Hilfsvariablen »X2« und »Y2« also die neuesten Koordinaten zugewiesen.

Der neuen Variablen »D2« wird der entsprechende Wert von V(RI) zugewiesen, so daß man also mit »D2« jetzt die Abweichung am Bildschirm angeben kann. Diese Anwendung sehen Sie auch gleich in Zeile 2900. Da soll die Farbe des zukünftigen Feldes bestimmt werden. Dafür übergibt man der Funktion als Argument einfach »D2«. Sollte die Farbe an der Stelle nicht Grün sein oder an der Stelle nicht das Zeichen mit dem POKE-Wert 160 stehen, wird nach Zeile 2980 verzweigt. Was heißt aber »das Zeichen mit dem POKE-Wert 160«? Ich habe das System mit den POKE-Werten bereits früher erklärt. Von damals sollten Sie noch wissen, daß die Zahl 32 für [SPACE] steht. Wenn Sie zu einem beliebigen POKE-Wert 128 addieren, erhalten Sie dasselbe Zeichen nur invers dargestellt. Bei unseren Golfkursen werden das Loch, Gras, Wasser, Sand, Blöcke etc. durch ein inverses [SPACE] dargestellt. Lediglich der Rand hat andere Werte. Die Farbe fünf steht für Grün. Sobald also die Farbe nicht mehr Grün ist bzw. an der neuen Stelle der Rand steht, verzweigt das Programm nach Zeile 2980.

Die Zeilen 2910 bis 2970 werden demnach nur bearbeitet, wenn sich Ihr Ball weiterhin auf Gras befindet. In dem Fall wird die alte Darstellung gelöscht (das heißt mit der ursprünglichen Farbe »BC$« überschrieben; 2910 bis 2920), und Ihr Ball an der neuen Position gesetzt (Zeilen 2930 bis 2950). Gleichzeitig wurden auch die Koordinaten »BX« und »BY« aktualisiert. In 2960 wird die Zeichenfarbe noch auf Grün gesetzt, da sich der Golfball weiterhin auf Gras befindet (siehe Bedingung in 2900). Ein Sprung nach 3640 beendet diesen Teil.

Ab Zeile 2980 kommt die Behandlung für alle anderen Fälle. Sollte die Farbe an der neuen Stelle null (für Schwarz) sein, aber kein inverses [SPACE] gesetzt sein, so heißt das, daß es sich nur noch um den Rand handeln kann (Zeile 2980). Für das Loch gilt nämlich immer noch BS(D2)=160!

Die Zeilen 3100 bis 3280 sind für den Rand zuständig. Bei Wasser (FA=6) treten die Zeilen ab 3290 in Aktion, bei Sand (FA=7) die ab 3381, bei Blöcken (FA=2) die ab 3400, und sollte das Loch erwischt worden sein (FA=0), springt der Computer nach 3520.

Doch zunächst kommen die Zeilen ab 3100. Hier steht, wie gesagt, die Behandlung für den Rand. Am Rand soll der Ball reflektiert werden. Wenn er senkrecht darauf fällt, ist das nicht weiter schwer (3100 und 3110). Bei RI=1 und RI=3 kann man jeweils vier addieren, während man bei RI=5 und RI=7 jeweils vier subtrahieren kann. Dann kann das Programm gleich wieder ab 2880 weiterarbeiten.

Bei den anderen Fällen wird es jedoch ein bißchen haarig. Zunächst unterscheidet das Programm zwischen den übrigen vier Fällen (ON RI/2 GOTO ..., Zeile 3120).

In einem solchen Fall dürfte eine kleine Tabelle ganz hilfreich sein:

Zeilen 3130 bis 3160:	RI=2
Zeilen 3170 bis 3200:	RI=4
Zeilen 3210 bis 3240:	RI=6
Zeilen 3250 bis 3280:	RI=8

»RI« gibt hier immer die RIchtung an, mit der der Ball angeflogen kommt. Prinzipiell gilt: Einfallswinkel = Ausfallswinkel. Doch muß man darauf achten, daß bei der neuen Richtung nicht gleich wieder ein Hindernis im Weg liegt. Darum wird jedesmal abgefragt, ob in der neuen Richtung BS(D2+-1) und BS(D2+-40) ein inverses [SPACE] steht. Das ist – wie gesagt – immer der Fall, mit Ausnahme des Randes; und nur am Rand wird reflektiert.

Ich glaube, es artet in zu trockene Theorie aus, wollte ich alle Befehle einzeln erklären. Daher mein Vorschlag: Wenn es Sie brennend interessiert, versuchen Sie, sich durchzuschlagen. Wenn Sie einen Teil verstanden haben, werden Sie auch den Rest verstehen, da alle Routinen ganz analog aufgebaut sind.

Die anderen kann ich beruhigen, daß es für Ihre Programmiererkarriere bestimmt nicht zwingend erforderlich ist, diesen Teil durchzuackern.

Ab Zeile 3290 kommt der Teil, der das Wasser behandelt. Ich sagte vorher, »TF« sei ein Flag für die Hindernisse. Das stimmt schon, aber es ist eigentlich noch ein bißchen mehr; es kann maximal den Wert drei annehmen, um das Hindernis nicht ewig lang bestehen zu lassen. Sehen wir uns das im einzelnen an:

Kommt der Computer zum ersten Mal nach 3290, ist »TF« noch null. Die Bedingung ist also nicht erfüllt, der Interpreter springt nach 3300, wo »TF« auf drei gesetzt wird. Gleichzeitig wird die Anzahl Ihrer Schläge um eins erhöht. Weiter unten (in 3640) werden Sie sehen, daß »TF« erniedrigt wird.

Auf alle Fälle ist »TF« ungleich null, sobald der Computer Zeile 3320 erreicht. Und dort wird die alte Darstellung gelöscht (mit der ursprünglichen Farbe überschrieben). Das kennen Sie mittlerweile schon. Entscheidend ist, daß »BC$« in 3370 auf blau gesetzt wird (für das Wasser).

Die Programmteile für Sand (Zeilen 3381 bis 3389), Unebenheiten (Zeilen 3400 bis 3510) und Blöcke (3520 bis 3630) arbeiten alle nach demselben Prinzip. Den Rest überlasse ich Ihnen. Das dürfte nicht mehr so schwer sein. Erinnern möchte ich nur daran, daß in »GS« die Geschwindigkeit Ihres Schlages gespeichert ist. Wenn diese also um »0.5« bzw. »0.3« verringert wird, sehen Sie ja, wie viele Stellen Sie mit welcher Kraft hinter sich bringen können.

18 Biorhythmus

Einführung

Man kann dieses Programm zwar nicht unbedingt Spiel nennen, aber dennoch braucht man eine kurze Anleitung dafür. Ob Sie dem Biorhythmus glauben oder nicht, bleibt Ihnen überlassen. Es gibt Leute, die den Biorhythmus zur Grundlage ihres täglichen Lebens machen, und es gibt Leute, die ihn »eher kritisch betrachten«, um es einmal vorsichtig zu formulieren. Ich finde den Biorhythmus zwar nicht besonders vertrauenswürdig, aber als SPIELerei ist er ganz lustig.

Nach dem Programmstart werden Sie nach Ihrem Namen gefragt, das macht das Ganze etwas persönlicher. Anschließend sollen Sie Ihr Geburtsdatum eingeben. Sie haben dabei zwei Möglichkeiten. Entweder tippen Sie jeweils nach den Tages- und Monatsangaben ein Komma, oder Sie drücken jeweils RETURN. Das sähe dann, wie folgt, aus:

```
IHR GEBURTSTAG (TT,MM,JJ)? 16,08,75
```

oder:

```
IHR GEBURTSTAG (TT,MM,JJ)? 05
?? 05
?? 05
```

Dabei gibt der Computer im zweiten Beispiel die doppelten Fragezeichen selbst aus. Beachten Sie bitte, daß Ihre Eingabe zweistellig erfolgen sollte. Das heißt nicht, daß Sie einstellige Zahlen immer mit einer vorangestellten Null eingeben müssen (wie oben dargestellt); aber Sie sollten keine drei- oder vierstellige Angaben (beispielsweise bei den Jahren) angeben. Da der Computer allerdings Ihre Eingabe überprüft, kann es praktisch zu keinen so groben Fehleingaben kommen.

Nachdem Sie Ihr Geburtsdatum eingegeben haben, werden Sie nach dem Startdatum der Kurve gefragt. Für die Eingabe gilt das oben Gesagte. Auch hier werden Fehleingaben abgefangen, denn was soll ein Biorhythmus vor dem Geburtsdatum? Solche Abfragen sollte man nicht als Gängelei des Benutzers ansehen, sondern als Sicherheit und Bedienerfreundlichkeit. Man kann den Computeranwender (da draußen vor der Mattscheibe) ja höflich darauf aufmerksam machen, daß er einen kleinen Fehler gemacht hat. Der Rest des Programms bedarf eigentlich keiner Erläuterung mehr. Lassen Sie sich überraschen!

Listing zu Biorhythmus

```
10 REM ******************
20 REM ***            ***
30 REM *** BIORHYTHMUS ***
40 REM ***            ***
50 REM ******************
60 REM
70 GOSUB 1ØØØ
8Ø GOSUB 2ØØØ
9Ø END
1ØØØ REM
1Ø1Ø REM *** VORBEREITUNGEN ***
1Ø2Ø REM
1Ø3Ø SW=29
1Ø4Ø DIM A$(SW), PL(23), EL(28), IL(33)
1Ø5Ø MO$="JANFEBMARAPRMAIJUNJULAUGSEPOKTNOVDEZ"
1Ø6Ø WO$="MODIMIDOFRSASO"
1Ø7Ø FOR I=1 TO SW
1Ø8Ø :A$(I)=" "
1Ø9Ø NEXT I
11ØØ B=INT (SW/2)
111Ø A=B+1.5
112Ø FOR I=1 TO 33
113Ø :IL(I)=INT (A+B*SIN(2*π*(I-1)/33))
114Ø :IF I>28 THEN NEXT I:GOTO 119Ø
115Ø :EL(I)=INT (A+B*SIN(2*π*(I-1)/28))
116Ø :IF I>23 THEN NEXT I:GOTO 119Ø
117Ø :PL(I)=INT (A+B*SIN(2*π*(I-1)/23))
118Ø NEXT I
119Ø BW$="-............Ø............+"
12ØØ RETURN
13ØØ REM
131Ø REM *** BERECHNUNG DES WOCHENTAGES
132Ø REM
133Ø IF M<3 THEN MØ=M+1Ø:JØ=J-1:GOTO 135Ø
134Ø MØ=M-2:JØ=J
135Ø Y=INT(JØ/1ØØ)
136Ø K=JØ-(Y*1ØØ)
137Ø N=INT((13*MØ-1)/5)+T+K+INT(K/4)+INT(Y/4)-2*Y+77
138Ø N=N-INT(N/7)*7+1
139Ø MØ=MØ-1
14ØØ Y=INT((146Ø97*Y)/4)+T+INT((1461*K)/4)+INT((153*MØ+2)/5)
141Ø RETURN
15ØØ REM
151Ø REM *** ERRECHNEN DER SCHALTJAHRE
152Ø REM
153Ø L=2
154Ø IF M<3 THEN L=Ø:GOTO 159Ø
155Ø IF J<>INT (J/4)*4 THEN 159Ø
156Ø IF J<>INT (J/1ØØ)*1ØØ THEN 158Ø
157Ø IF J<>INT (J/4ØØ)*4ØØ THEN 159Ø
158Ø L=1
159Ø Y1=INT ((3Ø55*(M+2))/1ØØ)-91+T-L
16ØØ RETURN
2ØØØ REM
2Ø1Ø REM *** "SPIEL" ***
2Ø2Ø REM
2Ø3Ø PRINT "{CLR,3DOWN}";TAB(11);"BIORHYTHMUS"
2Ø4Ø INPUT "{2DOWN}IHR NAME ";N$
2Ø5Ø INPUT "IHR GEBURTSTAG (TT,MM,JJ) ";T,M,J
2Ø6Ø IF T>Ø AND T<32 THEN GOTO 2Ø9Ø
2Ø7Ø PRINT "FALSCHER TAG !{2UP}"
2Ø8Ø GOTO 2Ø5Ø
```

```
2090 IF M>0 AND M<13 THEN GOTO 2120
2100 PRINT "FALSCHER MONAT !{2UP}"
2110 GOTO 2050
2120 IF J>1 AND J<100 THEN GOTO 2150
2130 PRINT "FALSCHES JAHR !{2UP}"
2140 GOTO 2050
2150 J=J+1900:MJ=J
2160 GOSUB 1300
2170 S=Y
2180 PRINT:PRINT "STARTDATUM DER KURVE ?"
2190 INPUT "(TT,MM,JJ) ";T1,M1,J1
2200 IF T1>0 AND T1<32 THEN GOTO 2230
2210 PRINT "FALSCHER TAG !{2UP}"
2220 GOTO 2190
2230 IF M1>0 AND M1<13 THEN GOTO 2260
2240 PRINT "FALSCHER MONAT !{2UP}"
2250 GOTO 2190
2260 IF J1>0 AND J1<100 THEN GOTO 2290
2270 PRINT "FALSCHES JAHR !{2UP}"
2280 GOTO 2190
2290 J1=J1+1900
2300 IF J1>J THEN GOTO 2350
2310 IF J1=J AND M1>M THEN GOTO 2350
2320 IF J1=J AND M1=M AND T1>=T THEN GOTO 2350
2330 PRINT "STARTDATUM LIEGT VOR DEM GEBURTSDATUM !{2UP}"
2340 GOTO 2190
2350 PRINT:INPUT "WIE VIELE TAGE ";Z
2360 IF Z<1 THEN GOTO 2350
2370 PRINT "{CLR}";TAB(14);"BIORHYTHMUS"
2380 PRINT TAB(14);"===FUER===="
2390 PRINT TAB((40-LEN(N$))/2);N$
2400 PRINT TAB(5);"GEBOREN AM ";
2410 PRINT MID$(WO$,2*N-1,2);T;MID$(MO$,3*M-2,3);MJ
2420 PRINT TAB(5);"P=PHYSISCH{6SPACE}(23 TAGE ZYKLUS)"
2430 PRINT TAB(5);"E=EMOTIONAL{5SPACE}(28 TAGE ZYKLUS)"
2440 PRINT TAB(5);"I=INTELLEKTUELL (33 TAGE ZYKLUS)"
2450 PRINT "{2DOWN}"
2460 M=M1:T=T1:J=J1:GOSUB 1300:GOSUB 1500
2470 Y2=Y1
2480 M=M+1:T=1:GOSUB 1500
2490 LE=Y1-Y2
2500 IF LE>Z THEN LE=Z
2510 Z=Z-LE
2520 T1=T1-1
2530 F=Y-S:E=F+LE-1
2540 FOR K=F TO E
2550 :T1=T1+1
2560 :PK=K-INT(K/23)*23+1
2570 :EK=K-INT(K/28)*28+1
2580 :IK=K-INT(K/33)*33+1
2590 :A$(PL(PK))="P"
2600 :IF A$(EL(EK))=" " THEN A$(EL(EK))="E":GOTO 2620
2610 :A$(EL(EK))="*"
2620 :IF A$(IL(IK))=" " THEN A$(IL(IK))="I":GOTO 2640
2630 :A$(IL(IK))="*"
2640 :IF A$(B+1)=" " THEN A$(B+1)="."
2650 :IF T1=1 OR K=F THEN PRINT MID$(C$,3*M1-2,3);J;ML$
2660 :PRINT MID$(W$,3*N-2,3);RIGHT$("{2SPACE}"+STR$(T1),3);"{2SPACE}";
2670 :FOR I=1 TO SW
2680 : PRINT A$(I);
2690 : A$(I)=" "
2700 :NEXT I
2710 :PRINT
2730 :GET A$:IF A$="" THEN GOTO 2730
```

```
2750 :N=N+1:IF N>7 THEN N=1
2760 NEXT K
2770 T1=1:M1=M1+1:IF M1>12 THEN M1=1:J1=J1+1
2780 IF Z THEN GOTO 2460
2790 PRINT TAB(9);ML$
2810 PRINT "MOECHTEN SIE EINE WEITERE KURVE?{SPACE,RVSON,SPACE,RVOFF}";
2820 GET AN$:IF AN$="" THEN GOTO 2820
2830 IF AN$="J" THEN GOTO 2000
2840 PRINT "{LEFT}N"
2850 RETURN
```

■ Programmbeschreibung für Biorhythmus

Ich befürchte, ich werde an diesem Programm nicht allzuviel erklären können. Der größte Teil davon besteht nämlich aus Rechnungen, die wohl niemand auswendig kann (am wenigsten ich). Jeder, der ein solches Programm schreiben will, wird wohl immer irgendwo nachschauen und Recherchen anstellen müssen, um an die Berechnungen des Wochentages, der Schaltjahre oder der einzelnen Kurven zu kommen.

Ich kann nur etwas über den Grobaufbau und das Programmtechnische sagen. Mal abwarten ...

Das Grundgerüst dieses Programms folgt natürlich treu »unserer« Linie (um nicht zu sagen, es ist linientreu). Damit meine ich den Aufbau mit den Unterroutinen ab 1000 und 2000. Der dritte und letzte Teil entfällt diesmal.

Zunächst werden vier eindimensionale Arrays definiert. »A$« ist später für die Ausgabe zuständig, während die anderen drei Variablen für die einzelnen »Rhythmen« zuständig sind: »P« für Physisch, »E« für Emotional und »I« für Intellektuell.

In den Klammern dahinter sehen Sie auch, wie lange ein »Rhythmus« (tut mir leid, mir fällt aber kein besseres Wort dafür ein, oder finden Sie Phase besser?) dauert. Ein physischer Zyklus dauert 23, ein emotionaler 28 und der intellektuelle 33 Tage.

In den nächsten beiden Variablen werden die MOnatsnamen bzw. WOchentage gespeichert. Auf diese beiden Variablen wird später noch zugegriffen. Die Variable »A$« (wie Ausgabe) muß zunächst einmal mit »SPACEs« vorbelegt werden, damit es später zu keinen falschen Aussagen kommt (das wäre ja unverzeihlich!). Und hier beginnen schon die ersten Berechnungen!

Das Unterprogramm ab Zeile 1300 ist wieder ein kurzer Lichtblick: Ich kann Ihnen wenigstens sagen, daß es für die Berechnung des Wochentages zuständig ist. Den Rest spare ich mir aber lieber – bevor Sie das Buch weglegen bzw. ich mich in Widersprüche verwickle.

Ab Zeile 1500 werden die Schaltjahre berechnet. Dabei ist es vielleicht ganz interessant zu wissen, daß die 100erjahre nur dann Schaltjahre sind, wenn sie nicht durch 400 teilbar sind. Sprich, das Jahr 2000 wird kein Schaltjahr, weil die Division 2000/400 wunderbar aufgeht.

Und dann beginnt schon der Programmteil »SPIEL«. Zu Beginn sehen Sie nur ganz profane Abfragen. Zunächst werden Sie nach Ihrem Geburtstag gefragt. Wie die Eingabe vonstatten geht, habe ich bereits in der Einführung zu diesem Spiel erklärt. In den Zeilen 2060 bis 2140 wird Ihre Eingabe erst einmal einer gründlichen Untersuchung unterzogen. Dabei wird getestet, ob Ihr GeburtsTag zulässig war. Das gleiche passiert mit dem GeburtsMonat und dem GeburtsJahr.

Da Ihre Eingabe zweistellig erwartet wird, muß zum Jahr noch 1900 addiert werden (ich möchte damit die älteren Leser unter Ihnen bestimmt nicht diskriminieren. Sollte für Sie noch das Geburtsjahrhundert 1800 zutreffen, bitte ich, das selbständig zu korrigieren!)

Nachdem Ihre Eingabe als richtig anerkannt wurde, kann der Computer Sie gleich mit der nächsten Frage nerven: Ab wann wollen Sie die Bio-Kurve?

Auch hier wird Ihre Eingabe genauestens untersucht, damit sich bei den Berechnungen auch keine Fehler einschleichen können. Diesmal kommt noch eine Überprüfung mehr dazu: Es muß getestet werden, ob das Startdatum vor Ihrem Geburtstag liegt. So eine Eingabe wäre sinnlos.

Aber auch hier muß ich mich noch einmal an die älteren Leser wenden: Entschuldigen Sie bitte auch hier meine Unverfrorenheit, einfach 1900 festzusetzen. Ich bitte vielmals um Entschuldigung ...

Die Ausgabe ab den Zeilen 2370 bis 2450 dürfte Ihnen klar sein. Nur eine Anmerkung zu Zeile 2390 sei mir gestattet: In »N$« ist Ihr Name (bzw. der, den Sie eingegeben haben) gespeichert. Das heißt, daß man mit LEN(N$) die Länge dieses Namens erhält. Das Ziel ist es nun, diesen Namen zentriert in der Mitte des Bildschirms ausgeben zu lassen.

Dazu muß man von der Mitte des Bildschirms (Spalte 20) die Hälfte der Namenslänge abziehen; dadurch erhält man die Startposition für die Ausgabe. Wenn man aber von 40 die gesamte Länge des Namens abzieht und anschließend halbiert, kommt man zum selben Ergebnis. Und das sehen Sie in Zeile 2390.

Über den Rest möchte ich keine großen Worte mehr verlieren. Die Zeilen, die vielleicht noch von allgemeinem Interesse sind (Zeilen 2590 bis 2630), sind nicht so schwer, als daß ich sie noch erklären müßte. Und der Rest ist – wie bereits gesagt – so fachspezifisch, daß ich es für unsinnig halte, die Formeln und Berechnungen genau darzulegen.

Ausklang

Jetzt haben Sie also alle Basic-Programme durchgearbeitet (sollten Sie jedenfalls). Ich hoffe, es hat ein bißchen Spaß gemacht, die Spiele haben Ihnen gefallen, und Sie haben eine Menge gelernt.

Wenn Sie jetzt eifrig drauf los programmieren, glaube ich, können Sie ziemlich schnell eine Menge Erfahrung sammeln. Und mit diesem Wissen und den Erfahrungen kann man Sie dann ruhig als fortgeschrittenen Programmierer bezeichnen; schließlich waren die Spiele alles andere als einfach.

Wenn ich sage, Sie sollten viel programmieren, dann meine ich damit nicht nur Spiele. Wagen Sie sich auch an andere Sachen heran. Aber Sie müssen sich einfach Ziele stecken, an die Sie dann auch ehrgeizig herangehen. Wie stünde es mit einer kleinen Adreßverwaltung für die eigene Familie? Achten Sie dabei zum Beispiel darauf, daß das Programm erweiterungsfähig ist. Die erste Version braucht ja nur die nötigsten Funktionen zu beherrschen. Aber wenn Sie mit der Zeit anspruchsvoller werden, kann es nie schaden, weitere Funktionen dazuzubasteln.

Lassen Sie sich auch nicht davon abhalten, wenn der Aufwand unverhältnismäßig groß zu sein scheint. Vom eigentlichen Nutzen her mag das ja stimmen, aber was Sie dabei an Erfahrung sammeln, sollte nicht unterschätzt werden. Vielleicht sind Sie auch in einem Verein, dem Sie anschließend anbieten, die Adreßverwaltung auf Ihrem Computer laufen zu lassen.

Aber es muß ja nicht unbedingt eine Adreßverwaltung sein. Wie stehts mit einem Mathematikprogramm für den Filius (oder, falls noch nicht vorhanden, für sich selbst)?

Programmideen gibt es wie Sand am Meer. Aber eine sollte man sich mindestens herauspicken, denn – wer rastet, der rostet. Das gilt auch für Basic-Kenntnisse.

Was ich Ihnen aber auch empfehlen kann, machen Sie so weiter wie bisher: Durch das Studieren fremder Programme kann man eine ganze Menge lernen. Für den Anfang ist das sogar immer sehr gut. Ich hoffe, Sie haben gemerkt, wie man es angeht, fremde Programme zu analysieren *und* zu verstehen!

Eine weitere Möglichkeit, die ich Ihnen ans Herz legen möchte, ist das regelmäßige Lesen von Computerzeitschriften. Zum einen sehen Sie dort viele fremde Programme, zum anderen können Sie da Tips von erfahrenen Programmierern bekommen. Und auch Ihr »Computer-Allgemeinwissen« kann dadurch beträchtlich erweitert werden.

Sie sehen, es gibt viele Möglichkeiten, sich »weiterzubilden«; nicht zuletzt gibt es auch noch weiterführende Basic-Bücher. Um Ihnen einen gewissen Anreiz dafür zu geben, sich nicht mit dem bisher Gelernten zufriedenzugeben, kommen hier noch zwei Programme. Sie sollen Ihnen den Mund wäßrig machen auf das, was man noch alles aus dem C64 herausholen kann.

Natürlich gibt es sehr viel beeindruckende Demonstrationen. Aber diese Programme sind auch in Basic gehalten; dadurch können Sie sehen, daß wirklich nur solche Befehle vorkommen, die Sie schon beherrschen, dennoch wird mit diesen Programmen ein völlig anderes (und nicht zu leichtes) Gebiet der Programmierung angesprochen. Wollte man sie daher erläutern, wäre es nötig, Ihnen die Grundkenntnisse zu vermitteln. Wie gesagt, wäre das ein vollkommen neues Gebiet, mit dem man ganze Bücher füllen kann. Daher wäre es hier als Abschluß dieses Buches denkbar ungeeignet, näher auf diese beiden »Spiele« einzugehen. Genießen Sie sie einfach!

Nach diesen zwei Programmen kommt noch ein Kapitel, das Ihnen zeigen soll, wie man Basic-Programme ein bißchen schneller machen kann. Teilweise können Sie das gleich in die Spiele aus diesem Buch einfließen lassen; aber auch das sind lediglich Anregungen. Die Tips sind ganz allgemein gehalten, orientieren sich also nicht an einem speziellen Spiel. Das soll das Verständnis fördern.

Die anschließenden Listen sind dazu gedacht, daß Sie dieses Buch hin und wieder als Nachschlagewerk in die Hand nehmen. Einige der Listen werden Sie jetzt noch nicht unbedingt verstehen. Sobald Sie sich aber auch über die Sprite- und Musikprogrammierung informiert haben, können Sie diese Tabellen sicher gut gebrauchen.

Programmoptimierung

Unter dieser Optimierung verstehe ich (zumindest in diesem Kapitel) nur die Beschleunigung von Programmen. Basic ist eine der langsamsten Sprachen. Ich habe Ihnen erklärt, daß es ein Programm im C64 gibt, das die Basic-Programme übersetzt, damit der Computer Ihre Befehle überhaupt versteht. Dieser Übersetzer (Interpreter) braucht natürlich seine Zeit, bis er die einzelnen Befehle erkennt, ausführt, den nächsten Befehl gefunden hat, erkennt, ausführt ...

Es gibt ein paar Sprachen (wie Pascal, C oder Modula), die zwar auch übersetzt werden, aber von einem Compiler. Da muß man das Programm ganz fertig haben, bevor man es dem Compiler zum Übersetzen gibt. Der macht daraus einen Code, den der Computer versteht. Und erst diesen Code kann man ausführen lassen. Es ist klar, daß diese Programme im Endeffekt wesentlich schneller arbeiten, da während des Programmlaufs die Übersetzung wegfällt. Andererseits ist es nicht gerade komfortabel, ein Programm erst vollständig erstellen zu müssen, bevor man es laufen lassen kann.

Um ein Maximum an Geschwindigkeit zu erreichen, muß man natürlich die »Muttersprache« des Computers verwenden, die sogenannte Maschinensprache (auch Assembler genannt). Keine andere Sprache kann in puncto Geschwindigkeit mit der Maschinensprache mithalten.

Aber wir sind ja hier, um Basic zu lernen. Und jetzt möchte ich Ihnen ein paar Tips zeigen. Dazu muß ich aber noch ein paar Sachen einführen: Ihr C64 hat eine Uhr eingebaut. Dafür gibt es zwei »Systemvariablen«, nämlich »TI$« und »TI«. Wenn Sie diese Uhr stellen wollen, müssen Sie »TI$« verwenden. Mit »TI« können Sie sich die Zeit ausgeben lassen.

Diese Uhr kann man hervorragend verwenden, wenn man die Laufzeit eines Programms ermitteln möchte. Sie können sich natürlich mit einer Stoppuhr neben den Computer setzen; aber damit werden Sie nie diese Genauigkeit erreichen. Wenn Sie also die Uhr zu Beginn eines Programms auf Null setzen:

```
10 TI$="000000"
```

und am Ende mit

```
1000 PRINT TI/60;"SEC."
```

können Sie sich die benötigte Laufzeit ausgeben lassen. Wenn Sie wollen, können Sie ja einmal eine kleine Schleife laufen lassen, um zu sehen, wie schnell der C64 arbeitet:

```
10 TI$="000000"
20 FOR I=1 TO 100:NEXT I
30 PRINT TI/60;"SEC."
```

Jetzt können wir zu den Tips schreiten: Ich glaube, ich habe oft genug gesagt, daß man Buchstaben auch mittels »POKE« auf den Bildschirm bringen kann. Es dürfte natürlich klar sein, daß man einen Text mit »PRINT« viel bequemer auf den Bildschirm bringen kann, als wenn man jeden einzelnen Buchstaben von Hand »POKEt«. Aber wie steht es mit der Geschwindigkeit?

```
10 TI="000000"
20 FOR I=1 TO 1000
30 :POKE 1023+I,1
40 NEXT I
50 PRINT TI/60;"SEC."
```

Ich werde hier nicht auf die genauen Ergebnisse der Laufzeiten eingehen. Aber lassen Sie jetzt einmal folgendes Programm laufen:

```
10 TI="000000"
20 FOR I=1 TO 1000
30 :PRINT "A";
40 NEXT I
50 PRINT TI/60;"SEC."
```

Ist der Unterschied deutlich geworden? Damit dürfte überhaupt klargeworden sein, daß der POKE-Befehl einer der langsamsten überhaupt ist. Aber es gibt noch weitere Möglichkeiten. Entfernen Sie beispielsweise den Doppelpunkt aus Zeile 30.

Um die Ergebnisse vergleichen zu können, möchte ich Sie bitten, vor jedem Programmstart den Bildschirm mit [SHIFT]+[CLR/HOME] zu löschen.

Überrascht Sie das? Der Doppelpunkt ist nur unnötiger Ballast. Das kann man an einem ganz einfachen Beispiel noch stärker verdeutlichen: Ersetzen Sie Zeile 30 durch

```
30 :PRINT "A";:REM 1000 A AUSGEBEN
```

Auch diese Mitteilung ist nur für den Benutzer, der Computer überliest diese Meldung; aber dafür braucht er eben seine Zeit. Und wenn er das 1000mal machen muß, dann summiert sich das. Jetzt löschen Sie bitte wieder diese Bemerkung und den Doppelpunkt. Das Programm ist gleich wieder schneller! Jetzt geht es aber noch weiter: Entfernen Sie bitte das »I« hinter dem NEXT.

Die Verbesserungen sind zwar nicht umwerfend, aber Sie merken, was man alles tun kann. Jetzt stellen Sie sich den Unterschied vor, wenn Sie zwei ineinandergeschachtelte Schleifen einmal mit und einmal ohne »I« (oder was auch immer) hinter dem »NEXT« laufen lassen. Wenn die Schleifen nur oft genug durchlaufen werden, kann das ganz schöne Zeitdifferenzen ergeben.

Jetzt setzen Sie aber mal das »NEXT« hinter den PRINT-Befehl (durch einen Doppelpunkt getrennt); Zeile 40 müssen Sie löschen.

```
30 PRINT "A";:NEXT
```

Das war nochmal schneller. Und jetzt entfernen Sie bitte das [SPACE] zwischen PRINT und den Anführungszeichen. Hätten Sie das gedacht?

Noch ein Trick:

```
20 FOR I=1 TO 1000:PRINT"A";:NEXT
```

Zeile 30 löschen! Und wieder schneller. Die Verbesserungen bewegen sich zwar mittlerweile im Zehntelsekunden-Bereich, aber sie sind nicht zu verachten.

Wenn Sie die SPACEs noch löschen, bringt das herzlich wenig bis gar nichts mehr. Aber es ist doch beeindruckend, was man alles machen kann. Gegen Ende dieses Kapitels möchte ich nur noch ein paar Tips geben, die Sie selber ausprobieren können.

`IF .... GOTO` ist schneller als `IF .... THEN`

Sie sehen also, Sie können entweder »GOTO« oder »THEN« weglassen und müssen nicht immer beide Befehle setzen, wie ich es in den Basic-Programmen praktiziert habe. Wenn Sie in einer IF-Abfrage ein »AND« haben, ist es besser, anstelle von:

```
IF ... AND .... THEN
```

zu schreiben:

```
IF ... THEN IF ....
```

Bei der ersten Version werden nämlich erst alle Bedingungen ausgerechnet; dagegen wird bei der zweiten Version bereits abgebrochen, wenn die erste Bedingung nicht erfüllt ist. Und wenn Sie die IF-Abfragen dann noch geschickt plazieren, können Sie noch mehr daraus machen.

Variablennamen sollten möglichst nur einen Buchstaben haben. Der Computer merkt sich nur die ersten zwei Stellen des Variablennamens, so daß längere Namen überflüssig sind. Aber einstellige sind immer noch schneller als zweistellige.

Wenn Sie eine Variable sehr oft benutzen, empfiehlt es sich, sie am Anfang des Programms einfach zu initialisieren, wenn es sein muß mit irgendeinem unschädlichen Wert. Das hat zur Folge, daß die Variable weiter vorne im Speicher plaziert wird, und der Computer sie dann schneller findet.

Bei großen Programmen sollten oft genutzte Unterroutinen am Anfang des Programms stehen. Und zwar sucht der Computer jedesmal, wenn er auf ein »GOSUB« trifft, das gesamte Programm von vorne nach hinten durch, bis er auf die angegebene Zeilennummer trifft. Und wenn diese möglichst weit vorne steht, wird er um so schneller fündig.

Ein letzter Tip, mit dem Sie aber vorsichtig umgehen sollten ist, ein Programm so dicht wie irgendmöglich zu schreiben. Das heißt, praktisch keine SPACEs drinlassen, möglichst viele Befehle in eine Zeile quetschen, möglichst wenig REMs einfügen.

Dazu möchte ich aber noch etwas sagen: Sie sollten ein Programm erst einmal auf Übersichtlichkeit hin anlegen. Wenn das Programm sorgfältig ausgetestet wurde, und sich kein Fehler ergab, dann können Sie daran gehen und das Programm komprimieren. Aber tun Sie das immer nur an einer Kopie, nie am Original. Wenn Sie nämlich einmal etwas ändern wollen, haben Sie bald keine Chance mehr, das Programm (das eigene!) auch nur zu verstehen, geschweige denn zu verbessern oder zu erweitern.

In diesem Buch konnte auf REMarks verzichtet werden, da ausführliche Beschreibungen die Programmzeilen gut dokumentierten. Aber ich nehme nicht an, daß Sie zu jedem einzelnen Programm einen halben Roman schreiben, um es später noch korrigieren zu können.

Denken Sie an meine Warnung. Spätestens, wenn Sie es das erste Mal falsch gemacht haben und es hinterher bereuen, werden Sie an meine Worte denken. Ich wünsche es Ihnen bestimmt nicht!

Und denken Sie daran: Geschwindigkeit ist nicht das Ein und Alles. Wenn Sie so auf Geschwindigkeit aus sind, kann ich Ihnen nur empfehlen, Maschinensprache zu lernen (das wird harte Arbeit, das verspreche ich Ihnen). Diese Tips sind vielmehr in kleinen Maßen zu genießen. Sie können sie beispielsweise bei zeitkritischen Programmteilen verwenden, aber auch hier gilt:

NUR AN EINER KOPIE AUSPROBIEREN!

Mit diesem Kapitel möchte ich mich von Ihnen verabschieden. Ich hoffe, die Spiele haben Spaß gemacht, und Sie haben viel gelernt. Es würde mich natürlich freuen, wenn Sie später in Ihrer Programmiererlaufbahn einmal an dieses Buch denken, und es als Nachschlagewerk benutzen (vor allem den Anhang). Auf alle Fälle wünsche ich alles Gute und viel Erfolg!

Anhang A

Basic-Befehle und Funktionen

ABS ()

Ergibt den absoluten Wert einer Zahl oder Variablen.
PRINT ABS (-3) ergibt 3

AND

Logischer Operator (zum Beispiel bei IF-Abfragen)
IF 3=5 AND "REGEN"="SONNE" THEN PRINT "WELTUNTERGANG"
Kann sowohl für IF-Abfragen als auch für binäre Operationen gebraucht werden.
IF A$="SONNENUNTERGANG" AND B$="EINSAM" THEN PRINT "TRAURIG"

ASC ()

Ergibt den ASCII-Code eines Zeichens.
PRINT ASC("A") ergibt 65.

ATN ()

Berechnet den ArcusTaNgens des Arguments (beachten Sie bitte das Bogenmaß!).
PRINT ATN(100) ergibt 1.56079666.

CHR$ ()

Ergibt das Zeichen mit dem ASCII-Code, der übergeben wird.
PRINT CHR$(65) ergibt »A«.
PRINT CHR$(ASC("A")) ergibt »A«.

CLOSE

Schließt die Kanäle zu Peripheriegeräten (schließt OPEN ab).
CLOSE 1

CLR

Setzt alle Variablen auf Null.
CLR

CMD

Leitet die Ausgabe auf einen vorher geöffneten Kanal um.
OPEN 1,4:CMD 1 (:CLOSE 1)

CONT

Fortsetzung eines Programmes nach einer STOP-Anweisung (oder Drücken der [RUN/STOP]-Taste).
CONT

COS ()

Ergibt den COSinus einer Zahl (beachten Sie bitte das Bogenmaß!).
PRINT COS(0) ergibt »1«

DATA

Daten, die mit »READ« einer Variablen zugeordnet werden.
DATA 1, 2, 3, 4, 5, 6, "GRUESS GOTT"

DEF FN

Sie können damit eigene Funktionen definieren.
DEF FN QU(X)=X*X

DIM

Dimensioniert ein Array.
DIM A$(100), B(450), C%(300)

END

Beendet ein Programm.
END

EXP ()

Berechnet »e« hoch Argument.
PRINT EXP(1) ergibt »e«

FOR

Definiert Start und Abbruchbedingung für eine Schleife.
FOR I=1 TO 25

FRE ()

Zeigt freien Speicherplatz an.
PRINT FRE(0)

GET

Weist einer Variablen den Inhalt des Tastaturpuffers zu, daher müssen Leereingaben abgefangen werden!
GET A$:IF A$="" THEN GOTO ...

GET#

Liest ein Zeichen von einem mit »OPEN« geöffneten Kanal.
OPEN 1,8,0,"KOENIG": GET#1,A$

GOSUB

Springt zu einem Unterprogramm ab der angegebenen Zeilennummer und kehrt später wieder hierher zurück.
GOSUB 200

GOTO

Verzweigt zur angegebenen Zeilennummer.
GOTO 200

IF/THEN

Testet eine Bedingung; falls die Bedingung erfüllt ist, wird der Teil hinter THEN abgearbeitet.
IF 3=3 THEN PRINT "GLEICH!"

INPUT

Wartet auf eine Eingabe des Benutzers, bis die [RETURN]-Taste gedrückt wurde.
INPUT "WIE GEHT ES IHNEN";B$

INPUT#

Liest Daten von einem mit »OPEN« geöffneten Kanal (bis zum nächsten [RETURN]).
OPEN 1,8,0,"KOENIG": INPUT#1, A$

INT ()

Schneidet die Nachkommastellen ab.
PRINT INT(-3.6) ergibt »4«

LEFT$ (,)

Trennt von einer Variablen eine bestimmte Zahl von Zeichen ab.
PRINT LEFT$("GUTEN MORGEN",3) ergibt »GUT«.

LEN ()

Ermittelt die Länge eines Strings.
PRINT LEN("AUF WIEDERSEHEN") ergibt »15«

LET

Weist einer Variablen den Wert eines Ausdrucks zu.
LET X=X1+X2

LIST

Gibt ein im Speicher stehendes Programm auf den Bildschirm aus.
LIST
LIST ZNUMMER -
LIST - ZNUMMER
LIST ZNUMMER - ZNUMMER

LOAD

Lädt Programme vom angegebenen Peripheriegerät.
LOAD "PROGRAMMNAME",8 lädt »PROGRAMMNAME« von Diskette
LOAD "$",8 lädt das Directory der Diskette

LOG ()

Berechnet den LOGarithmus zur Basis »e«.
PRINT LOG (2.71828182818) ergibt »1«

MID$ (,s,l)

Trennt von einem String »l« Zeichen ab der s.Position ab.
PRINT MID$("GRUESS GOTT",3,4) ergibt »UESS«

NEW

Löscht ein im Speicher befindliches Programm.
NEW

NEXT

Schließt eine FOR-Schleife ab.
FOR I=1 TO 25:PRINT "HALLO!":NEXT I

NOT

Logische Verneinung (dreht praktisch alles um).
»IF NOT (3=4) THEN PRINT "GLEICH"« entspricht »IF 3<>4 THEN PRINT "GLEICH"«

Dies ist wieder ein Befehl, der sowohl in IF-Abfragen als auch bei Binärarithmetik verwendet werden kann.

IF NOT A$="SONNENUNTERGANG" THEN PRINT "SONNENAUFGANG"

ON

Für berechnete Sprünge innerhalb eines Programms.
ON A GOTO 300, 400, 500, 600
ON B GOSUB 700, 800, 900, 1000

OPEN

Öffnet einen Kanal bzw. eine Datei zur Bearbeitung (wird mit »CLOSE« geschlossen).
OPEN 1,8,0,"KOENIG" öffnet Datei
OPEN 2,4,7 öffnet Kanal zum Drucker

OR

Logisches ODER in einer IF-Abfrage.
IF 3=5 OR "REGEN"="SONNE" THEN PRINT "WELTUNTERGANG"
Dieser Befehl kann – wie AND – sowohl in IF-Abfragen als auch für Binärarithmetik verwendet werden.
IF A$="ROT" OR A$="GELB" THEN PRINT "DIE AMPEL ZEIGT KEIN GRÜN!"

PEEK ()

Liest den Speicherinhalt an der angegebenen Adresse.
PRINT PEEK(1024)

POKE ,

Schreibt an die angegebene Speicheradresse den angegebenen Wert.
POKE 1024,1

POS ()

Ermittelt die X-Position des Cursors in einer Zeile. Das Argument wird nicht benutzt und kann alle Werte annehmen.
PRINT POS(0)

PRINT

Gibt Zeichenketten, Zahlen und Variablen auf den Bildschirm aus.
PRINT "GRUESS GOTT"

PRINT#

Gibt Zeichenketten etc. auf einen mit »OPEN« geöffneten Kanal beziehungsweise auf eine geöffnete Datei aus.
OPEN 1,4: PRINT#1,"GRUESS GOTT"

READ

Liest Daten aus einer DATA-Zeile und ordnet sie einer Variablen zu.
READ A, B$
DATA 3, "GUT"

REM

Beginnt einen Kommentar.
REM KOMMENTAR

RESTORE

Setzt den internen Zeiger auf das erste DATA-Element, das mit »READ« gelesen werden soll.
RESTORE

RETURN

Schließt ein Unterprogramm ab, das mit »GOSUB« aufgerufen wurde. Das Programm kehrt zu der Zeile zurück, von der aus das Unterprogramm aufgerufen wurde.
RETURN

RIGHT$ (,)

Trennt den angegebenen rechten Teil einer Variablen ab.
PRINT RIGHT$("GRUESS GOTT",4) ergibt "GOTT"

RND ()

Ermittelt Zufallszahlen zwischen null und eins.
PRINT INT (RND(1)*10)+1 ergibt eine Zufallszahl zwischen eins und zehn.

RUN

Startet ein im Speicher befindliches Basic-Programm.
RUN
RUN 100 Das Programm soll erst bei Zeile 100 abgearbeitet werden.

SAVE

Speichert ein Programm auf dem angegebenen Peripheriegerät.
SAVE "PROGRAMMNAME",8 speichert das Programm unter »PROGRAMMNAME« auf die Diskette.

SGN ()

Gibt das Vorzeichen eines Arguments aus.
a=SGN(b)

SIN ()

Berechnet den SINus einer Zahl (bitte beachten Sie das Bogenmaß!).
PRINT SIN(3.1415926535) ergibt »0«.

SPC ()

Druckt die angegebene Zahl von SPACEs aus.
PRINT SPC(10) gibt zehn Leerstellen aus.

SQR ()

Berechnet die Quadratwurzel einer Zahl.
PRINT SQR(64) ergibt »8«.

STOP

Bricht ein Programm ab. Das Programm kann anschließend mit »CONT« wieder fortgesetzt werden.
STOP

STR$ ()

Wandelt eine Zahl in eine Zeichenkette um.
A$ = STR$(145)

SYS

Ruft ein Maschinenspracheprogramm auf, das ab der angegebenen Adresse steht.
SYS 58640

TAB ()

Setzt den Cursor an die angegebene Spaltenposition.
PRINT TAB(10);"HALLO"

TAN ()

Berechnet den TANgens eines Winkels. (Bitte beachten Sie das Bogenmaß!)
PRINT TAN (3.1415926535)

USR ()

Startet ein Maschinenprogramm, dessen Startadresse in den Speicheradressen »85« und »86« abgelegt ist.
Y=USR(Z)

VAL ()

Wandelt eine Zeichenkette in einen numerischen Wert um.
B = VAL("145")

VERIFY

Vergleicht ein Programm mit dem im Speicher befindlichen.
VERIFY "TEST",8

WAIT

Hält ein Programm so lange an, bis eine Speicherstelle den gewünschten Wert erhält.
WAIT 198,1

Anhang B

Fehlermeldungen

BAD DATA

Es wurde versucht, aus einem File Zeichenketten einzulesen. Vom Programm wurden jedoch numerische Werte erwartet.
GET#1, A, B, C
Im File steht aber »GRUESS«, »GOTT«, »TSCHUESS!«

BAD SUBSCRIPT

Sie haben bei einem Array auf ein Feldelement zugegriffen, das es laut DIM-Befehl gar nicht gibt.
DIM A(14)
PRINT A(23)

BREAK

Das aktuelle Programm wurde durch Drücken von [RUN/STOP] unterbrochen.

CAN'T CONTINUE

Der Befehl »CONT« kann nicht ausgeführt werden, wenn Sie das Programm nicht mit »RUN« gestartet haben (sondern vielleicht mit GOTO), oder wenn Sie nach dem Programmabbruch Zeilennummern etc. verändert haben.

DEVICE NOT PRESENT

Entweder haben Sie versucht, ein Peripheriegerät (Floppy oder Drucker) anzusprechen, das nicht eingeschaltet war, oder das gar nicht existiert (zum Beispiel SAVE "IRGENDWAS",11).

DIVISION BY ZERO

Eine Division durch null ist nicht erlaubt, da sie mathematisch nicht definiert ist.

EXTRA IGNORED

Wenn Sie bei einer INPUT-Anweisung unerlaubte Zeichen (»,«, »:«) eingeben, erfolgt diese Fehlermeldung; doch der C64 setzt das Programm fort.

FILE NOT FOUND

Sie haben versucht, auf ein File zuzugreifen, das nicht existiert, oder dessen Name anders geschrieben wird (vielleicht ein Tippfehler?).

FILE NOT OPEN

Es wurde versucht, einen Kanal zu benutzen, der zuvor nicht mit »OPEN« geöffnet wurde.

FILE OPEN

Wenn Sie einen Kanal ein zweites Mal öffnen wollen, ohne ihn vorher geschlossen zu haben, tritt diese Fehlermeldung auf.

FORMULA TOO COMPLEX

Ein Ausdruck ist zu komplex, oder eine Berechnung enthält zu viele Klammern.

ILLEGAL DIRECT

Ein Befehl, der nur im Programmodus arbeitet, wurde im Direktmodus eingegeben (zum Beispiel INPUT).

ILLEGAL DEVICE NUMBER

Es wurde eine Geräteadresse über 15 oder eine unzulässige Geräteadresse angegeben.

ILLEGAL QUANTITY

Wenn Sie einer Funktion ein Argument mitteilen, das außerhalb des gültigen Bereichs liegt, beschwert sich der C64.

LOAD

Beim Laden eines Programmes ergaben sich Schwierigkeiten.

MISSING FILE NAME

Sie haben vergessen, bei einem Befehl den Filenamen anzugeben, z.B. »LOAD "",8«.

NEXT WITHOUT FOR

Der Interpreter findet zu einem NEXT-Befehl nicht das dazugehörige »FOR«.

```
FOR I=1 TO 23:NEXT O
```

Dieser Fehler tritt leicht bei verschachtelten Schleifen auf:

```
FOR I=1 TO 10
:FOR J=3 TO 6
: PRINT "HALLO"
:NEXT I
NEXT J
```

NOT INPUT FILE

Sie wollten aus einem File, das Sie nur zum Schreiben geöffnet hatten, Daten einlesen.

NOT OUTPUT FILE

Diesmal hatten Sie ein File zum Lesen geöffnet und wollten Daten in das File schreiben.

OUT OF DATA

Sie wollten mehr Daten mittels »READ« aus den DATA-Zeilen auslesen als vorhanden waren.

OUT OF MEMORY

Sie haben keinen Speicherplatz mehr zur Verfügung. Dieser Fehler tritt zum Beispiel auf, wenn Sie riesige Variablenmengen zu verarbeiten haben oder zu viele Schleifen verschachtelt haben. Wenn Sie ein »CLR« verschmerzen können, probieren Sie es einmal damit, oder speichern Ihr Programm ab, schalten den C64 aus und wieder an und laden es noch einmal. Dieser Fehler kann nämlich auch dann auftreten, wenn Sie in einem großen Programm viele Zeilen geändert haben.

OVERFLOW

Ihre Berechnungen sprengen den Rahmen des C64. Die (auf dem C64) darstellbaren Zahlen liegen zwischen:
2.93873588E -39
1.70141183E +38
E steht für »mal zehn hoch«.

REDIM'D ARRAY

Sie wollten ein Feld zum zweiten Mal dimensionieren.

REDO FROM START

Sollten Sie bei einer INPUT-Anweisung Buchstaben eingegeben haben, obwohl Zahlen erwartet wurden, erscheint diese Fehlermeldung, und Sie bekommen eine zweite Chance.

RETURN WITHOUT GOSUB

Der C64 stieß auf ein »RETURN«, ohne vorher ein entsprechendes »GOSUB« gefunden zu haben.

STRING TOO LONG

Sie wollten einem String mehr als 255 Zeichen zuordnen.

SYNTAX

Diesem Fehler dürften Sie am häufigsten über den Weg laufen. Sobald der Interpreter einen Befehl nicht erkennen kann, gibt er diesen Fehler aus. Das kann an einer fehlenden Klammer, an einem Tippfehler etc. liegen.

TOO MANY FILES

Es wurde versucht, eine Datei zu öffnen, obwohl schon zehn Dateien geöffnet waren.

TYPE MISMATCH

Sie wollten einem String eine Zahl zuordnen oder umgekehrt. Dieser Fehler tritt auch auf, wenn Sie andere mathematische Symbole (wie »–«, »*«,»/«) für zwei Strings benutzen (»+« ist erlaubt!).

UNDEF'D FUNCTION

Sie wollten mit »FN« eine Funktion aufrufen, die noch nicht mit »DEF FN« definiert worden war.

UNDEF'D STATEMENT

Ein GOTO bzw. GOSUB sollte zu einer Zeile verzweigen, die nicht existiert.

VERIFY

Wenn Sie nach dem Speichern den Befehl »VERIFY« ausführen lassen, und die Daten auf der Diskette nicht mit denen im Speicher übereinstimmen, kommt es zu diesem Fehler.

Anhang C

Werte für FarbPOKEs

0	Schwarz	8	Orange
1	Weiß	9	Braun
2	Rot	10	Hellrot
3	Türkis	11	Grau 1
4	Violett	12	Grau 2
5	Grün	13	Hellgrün
6	Blau	14	Hellblau
7	Gelb	15	Grau 3

Wichtige Speicherstellen:

197	Taste gedrückt?
646	Zeichenfarbe (Farbe des Cursors)
650	Tastenwiederholung
53280	Rahmenfarbe
53281	Hintergrundfarbe

Anhang D

Speichertabelle für Musik

SI=54272

1. Stimme

SI	Frequenz Lowbyte	0 bis 255
SI+1	Frequenz Highbyte	0 bis 255
SI+2	Pulsbreite Lowbyte	nur bei Rechteck: bis 225
SI+3	Pulsbreite Highbyte	nur bei Rechteck: bis 15
SI+4	Wellenform	Rauschen 129
		Rechteck 65
		Sägezahn 33
		Dreieck 17
SI+5	Anschlag abschwellen	16∗AN+AB (AN, AB < 16)
SI+6	Halten/Ausklingen	16∗AS+AF (AS, AF < 16)

2. Stimme

SI+7	Frequenz Lowbyte	0 bis 255
SI+8	Frequenz Highbyte	0 bis 255
SI+9	Pulsbreite Lowbyte	nur bei Rechteck: bis 225
SI+10	Pulsbreite Highbyte	nur bei Rechteck: bis 15
SI+11	Wellenform	Rauschen 129
		Rechteck 65
		Sägezahn 33
		Dreieck 17
SI+12	Anschlag abschwellen	16∗AN+AB (AN, AB < 16)
SI+13	Halten/Ausklingen	16∗AS+AF (AS, AF < 16)

3. Stimme

SI+14	Frequenz Lowbyte	0 bis 255
SI+15	Frequenz Highbyte	0 bis 255
SI+16	Pulsbreite Lowbyte	nur bei Rechteck: bis 225
SI+17	Pulsbreite Highbyte	nur bei Rechteck: bis 15
SI+18	Wellenform	Rauschen 129
		Rechteck 65
		Sägezahn 33
		Dreieck 17
SI+19	Anschlag abschwellen	16∗AN+AB (AN, AB < 16)
SI+20	Halten/Ausklingen	16∗AS+AF (AS, AF < 16)
SI+21	Filterfrequenz Lowbyte	
SI+22	Filterfrequenz Highbyte	
SI+23	Stimme 1: 1	
Stimme 2	2	
Stimme 3	4	
Extern	8	
Resonanz		16, 32, 64, 128
SI+24	Lautstärke für Stimmen eins bis drei (von 0 bis 15)	
Tiefpaß	16	
Bandpaß	32	
Hochpaß	64	
Stimme 3		
aus	128	

Anhang E

Frequenztabelle

Nummer	Note/Oktave	Frequenz dezimal	HI	LOW
0	C-0	268	1	12
1	C#-0	284	1	28
2	D-0	301	1	45
3	D#-0	318	1	62
4	E-0	337	1	81
5	F-0	358	1	102
6	F#-0	379	1	123
7	G-0	401	1	145
8	G#-0	425	1	169
9	A-0	451	1	195
10	A#-0	477	1	221
11	H-0	506	1	250
16	C-1	536	2	24
17	C#-1	568	2	56
18	D-1	602	2	90
19	D#-1	637	2	125
20	E-1	675	2	163
21	F-1	716	2	204
22	F#-1	758	2	246
23	G-1	803	3	35
24	G#-1	851	3	83
25	A-1	902	3	134
26	A#-1	955	3	187

Nummer	Note/Oktave	Frequenz dezimal	HI	LOW
27	H-1	1012	3	244
32	C-2	1072	4	48
33	C#-2	1136	4	112
34	D-2	1204	4	180
35	D#-2	1275	4	251
36	E-2	1351	5	71
37	F-2	1432	5	152
38	F#-2	1517	5	237
39	G-2	1607	6	71
40	G#-2	1703	6	167
41	A-2	1804	7	12
42	A#-2	1911	7	119
43	H-2	2025	7	233
48	C-3	2145	8	97
49	C#-3	2273	8	225
50	D-3	2408	9	104
51	D#-3	2551	9	247
52	E-3	2703	10	143
53	F-3	2864	11	48
54	F#-3	3034	11	218
55	G-3	3215	12	153
56	G#-3	3406	13	78
57	A-3	3608	14	12
58	A#-3	3823	14	239
59	H-3	4050	15	210
64	C-4	4291	16	195
65	C#-4	4547	17	195
66	D-4	4817	18	209
67	D#-4	5103	19	239
68	E-4	5407	21	31
69	F-4	5728	22	96
70	F#-4	6069	23	181
71	G-4	6430	25	30
72	G#-4	6812	26	156
73	A-4	7217	28	49
74	A#-4	7647	29	223

Nummer	Note/Oktave	Frequenz dezimal	HI	LOW
75	H-4	8101	31	165
80	C-5	8583	33	135
81	C#-5	9094	35	134
82	D-5	9634	37	162
83	D#-5	10207	39	226
84	E-5	10814	42	62
85	F-5	11457	44	193
86	F#-5	12139	47	107
87	G-5	12860	50	60
88	G#-5	13625	53	57
89	A-5	14435	56	99
90	A#-5	15294	59	190
91	H-5	16203	63	75
96	C-6	17167	67	15
97	C#-6	18188	71	12
98	D-6	19269	75	69
99	D#-6	20415	79	191
100	E-6	21629	84	125
101	F-6	22915	89	131
102	F#-6	24278	94	214
103	G-6	25721	100	121
104	G#-6	27251	106	115
105	A-6	28871	112	199
106	A#-6	30588	119	124
107	H-6	32407	126	151
112	C-7	34334	134	30
113	C#-7	36376	142	24
114	D-7	38539	150	139
115	D#-7	40803	159	126
116	E-7	43258	168	250
117	F-7	45830	179	6
118	F#-7	48556	189	172
119	G-7	51443	200	243
120	G#-7	54502	212	230
121	A-7	57743	225	143
122	A#-7	61176	238	248
123	H-7	64814	253	46

Anhang F

Speichertabelle für Sprites

V=53248

V+0	X-Koordinate für Sprite 0
V+1	Y-Koordinate für Sprite 0
V+2	X-Koordinate für Sprite 1
V+3	Y-Koordinate für Sprite 1
V+4	X-Koordinate für Sprite 2
V+5	Y-Koordinate für Sprite 2
V+6	X-Koordinate für Sprite 3
V+7	Y-Koordinate für Sprite 3
V+8	X-Koordinate für Sprite 4
V+9	Y-Koordinate für Sprite 4
V+10	X-Koordinate für Sprite 5
V+11	Y-Koordinate für Sprite 5
V+12	X-Koordinate für Sprite 6
V+13	Y-Koordinate für Sprite 6
V+14	X-Koordinate für Sprite 7
V+15	Y-Koordinate für Sprite 7
V+39	Sprite 0 : Farbe
V+40	Sprite 1 : Farbe
V+41	Sprite 2 : Farbe
V+42	Sprite 3 : Farbe
V+43	Sprite 4 : Farbe
V+44	Sprite 5 : Farbe
V+45	Sprite 6 : Farbe
V+46	Sprite 7 : Farbe

Für die folgenden Speicherstellen gilt diese Darstellung:

Sprite-Nr.:	7	6	5	4	3	2	1	0
Bit:	128	64	32	16	8	4	2	1

V+16	X-Koordinate über 255
V+21	Sprite ein/ausschalten
V+23	Sprite in Y-Richtung vergrößern
V+27	Hintergrund-Priorität
V+28	Sprite mehrfarbig
V+29	Sprite in X-Richtung vergrößern
V+30	Kollision mit Sprite
V+31	Hintergrund-Kollision

Stichwortverzeichnis

Axel Seibert

C64/C128 Spielend BASIC lernen

AXEL SEIBERT, geboren 1970, besucht das Karlsgymnasium in München. Schon seit 1984 beschäftigt er sich intensiv mit dem C64. Seine Fachgebiete sind Programmiersprachen und Spiele. 1988 kaufte er sich zusätzlich einen Amiga, um sich verstärkt mit den vielfältigen Möglichkeiten dieses Rechners auseinanderzusetzen.

Möchten Sie spielend Basic lernen? Wollen Sie dabei nicht nur graue Theorie vorgesetzt bekommen? Wollen Sie vielmehr ein Buch mit ansprechenden Programmen, die nicht erst umständlich abgetippt werden müssen? In diesem Fall ist »Spielend Basic lernen« das richtige Buch – eine leichtverständliche Einführung in das Basic des C64. Sie lernen spielerisch programmieren, das heißt, anhand von sehr ausführlich dokumentierten Spielprogrammen werden Ihnen Schritt für Schritt die Fähigkeiten und Möglichkeiten der Programmiersprache Basic gezeigt. Die Programme sind alle klar aufgebaut, beginnen leicht und werden mit der Zeit anspruchsvoller, so daß Basic-Befehle und Befehlsstrukturen in Einsatz und Anwendung leicht zu verstehen sind. So wird in »Farbendreher« eine Sortierroutine erklärt. Das unsichtbare »Labyrinth« ist schon schwieriger, denn Sie programmieren ein Spiel, bei dem der Spieler von einem Drachen verfolgt wird. Bei »Würmli« wird zuerst ein Irrgarten konstruiert, der dann von einer Spielfigur selbständig bewältigt wird. 21 lustige Spiele werden ausführlich in ihrer Programmierung vorgestellt und können beliebig verändert oder erweitert werden.

Im weiteren Verlauf werden verschiedene Programmiertechniken dargestellt und praktisch eingesetzt. Abgerundet wird der Programmierkurs mit Hinweisen und Tips, die den Programmlauf beschleunigen.

Besonders erleichtert wird Ihnen die Programmierarbeit durch die beigelegte Diskette. Auf ihr sind alle vorgestellten Programme enthalten, somit entfällt das fehlerträchtige Abtippen. Ebenso sind im Buch alle Listings abgedruckt, Sie benötigen also keinen Drucker, um ein Programm einmal vollständig zu überblicken.

Weiter finden Sie im Anhang tabellarische Aufstellungen über Basic-Befehle und Funktionen, Fehlermeldungen, Farb-Pokes, eine Speichertabelle für Musik, eine Frequenztabelle und eine Speichertabelle für Sprites.

Nach dem Studium des Buches sind Sie in der Lage, selbständig eigene Programme zu entwerfen und problemlos in Basic umzusetzen.

Die Begleitdiskette:

Auf ihr sind alle im Buch beschriebenen Spiele enthalten.

Hardware-Anforderungen:

C64 oder C128 im 64er-Modus mit einem Diskettenlaufwerk 1541/1570/1571.

ISBN 3-89090-701-6

03900

4 001057 907010

DM 39,– sFr 35,90 öS 304,–



